Breaking the $\Omega(|C|\kappa)$ Barrier on Garbled Circuit Size in the Random Oracle Model

Junru Li
Tsinghua University
jr-li24@mails.tsinghua.edu.cn

Yifan Song
Tsinghua University
and Shanghai Qi Zhi Institute
yfsong@mail.tsinghua.edu.cn

Abstract

In this paper, we study garbled circuits in the random oracle model against a computationally unbounded adversary with T queries to a (programmable) random oracle. From Yao's garbled circuits (SFCS 1986) to Three-Halves (CRYPTO 2021), the garbled circuit size has been reduced from $8|C|(\log T + \kappa)$ bits to $1.5|C|(\log T + \kappa)$ bits for achieving a statistical error of $2^{-\kappa}$, where $|C|$ is the circuit size and κ is the statistical security parameter. However, no known result achieves $o(|C|\kappa)$ bits of garbled circuit size by now, and it is widely believed that a garbled circuit must have $\Omega(|C|\kappa)$ bits in the random oracle model.

In this work, we present the first garbling scheme that achieves $o(|C|\kappa)$ bits of garbled circuit size in the random oracle model. In particular, for a circuit C of size $|C|$ and depth D, the achieved garbled circuit size is $O(|C|\log T + D\kappa^2 \log T)$ bits. This breaks the long-standing $\Omega(|C|\kappa)$ barrier on the garbled circuit size.

We extend our garbling scheme to a maliciously secure two-party computation protocol with communication of $O(|C|\log T + D(\log T + \kappa)^2 \log T + \mathsf{poly}(\kappa, \log T))$ bits against any T-query adversary assuming parallel oblivious transfers and a (programmable) random oracle. The protocol only requires 1 OT round and 3 one-way communication rounds. If only requiring one of the two parties to have output, a similar communication complexity can be achieved for constructing a non-interactive secure computation (NISC) protocol, which only relies on the preprocessing of bit-OT correlations and a random oracle. Compared to a concurrent work on NISC by Ishai et al. (EUROCRYPT 2026) in the same setting, we achieve a better amortized communication cost per gate at the cost of an additional term related to the circuit depth. The NISC protocol with a similar communication cost can also be constructed from a (slightly stronger version of) semi-malicious 2-round OT protocol.

Contents

1 Introduction **4**
- 1.1 Our Contribution . . . 4
- 1.2 Related Works . . . 7

2 Technical Overview **9**
- 2.1 Starting Idea: Multiparty Garbling Protocol in the Head . . . 9
- 2.2 Review of GLOS Multiparty Garbling Protocol . . . 9
- 2.3 Constructing Garbling Scheme with $o(|C|\kappa)$ GC size . . . 10
- 2.4 Maliciously Secure 2PC Protocol . . . 13

3 Preliminaries **16**
- 3.1 Security Model . . . 16
- 3.2 Parallel Oblivious Transfers . . . 16
- 3.3 Linear Secret Sharing Schemes . . . 16
- 3.4 Garbling Schemes . . . 16
- 3.5 Chernoff Bound . . . 17

4 The Underlying Protocol **18**

5 Main Garbling Scheme **18**
- 5.1 The Garbling Algorithm . . . 19
- 5.2 The Encoding Algorithm . . . 20
- 5.3 The Evaluation Algorithm . . . 21
- 5.4 The Decoding Algorithm . . . 22

6 Maliciously Secure 2PC **22**

A An Overview for [GLOS25] **32**

B Formal Definitions of Linear Secret Sharing Schemes **33**

C The Formal Security Requirement for the Underlying Protocol **34**

D An Instantiation of the Underlying Protocol **34**
- D.1 Subprotocols for Π_0 . . . 34
- D.2 Preprocessing and Input Functionalities . . . 38
- D.3 Protocol Description . . . 38
- D.4 Security Proof . . . 40
- D.5 Towards Underlying Protocol for Malicious 2PC . . . 46

E Instantiation of the Linear Secret Sharing Scheme **46**
- E.1 Basic Algebraic Geometry . . . 46
- E.2 The Linear Secret Sharing Scheme . . . 47
- E.3 Error Correction . . . 49

F Security Proof for the Garbling Scheme **49**
- F.1 Proof of Privacy . . . 50
- F.2 Proof of Obliviousness . . . 54
- F.3 Proof of Authenticity . . . 54

G Cost Analysis for the Garbling Scheme **54**

H Detailed Evaluation Process in Our 2PC **55**

I Security Proof for the 2PC **56**
I.1 Security against a Corrupted Garbler 56
I.2 Security against a Corrupted Evaluator 61

J Cost Analysis for the 2PC **67**

K Towards NISC **68**
K.1 Brief Overview of the Construction 68
K.2 Construction from Bit-OT Correlations 68
K.3 Security Proof for the Protocol from Random Bit-OT Correlations 72
K.4 Construction from 2-Round OT Protocols 84
K.5 Security Proof for the Protocol from 2-Round OTs 89

1 Introduction

Garbled circuits (GC), introduced by Yao [Yao86] and later formalized by [BHR12] using the notion of garbling schemes, serve as an important building block in constructing secure multiparty computation (MPC) protocols [Yao82]. In particular, a garbled circuit directly implies a constant-round semi-honest 2PC protocol if further assuming oblivious transfers (OTs) [Yao86], and it is also the main technique in constructing efficient constant-round MPCs in various settings [YWZ20, BGH+23, GLM+24, HKN+25, GLOS25, LS25a, LS25b]. To construct communication-efficient protocols, the main bottleneck is the size of a garbled circuit. More concretely, Yao's garbled circuit [Yao86] is 8κ times larger than the original circuit only relying on one-way functions, where κ is the security parameter. After that, a long line of works [BMR90, NPS99, PSSW09, KS08, KMR14, ZRE15, GLNP15, RR21] focus on reducing the garbled circuit size using symmetric cryptographic primitives, and the state-of-the-art work [RR21] achieves a garbled circuit of size 1.5κ bits per AND gate with free garbling for XOR gates (freeXOR).

The $\Omega(|C|\kappa)$ Barrier on GC Size. It is widely believed that the garbled circuit size is at least $\Omega(\kappa)$ bits per gate with only symmetric-key operations[1] with security error proportional to $2^{-\kappa}$. As we have introduced, the best-known garbled circuit in this setting remains to be $\Omega(|C|\kappa)$ bits for over 35 years! There are also works on proving the $\Omega(\kappa)$-bit lower bound for garbling a single gate in various restricted models [ZRE15, BK24, FLZ24, XHX25, JRR25] (see Section 1.2 for a more detailed discussion). These lower bounds are applied in the random oracle model (ROM), where the adversary is computationally unbounded but with a bounded number of queries to a random oracle. These lower bounds provide evidence on the conjecture that a garbled circuit requires at least $\Omega(|C|\kappa)$ bits.

We note that one can artificially reduce the garbled circuit size to $\Omega(|C| \cdot \omega(\log \kappa))$ by sacrificing the security error from $2^{-\kappa}$ to $2^{-\omega(\log \kappa)}$, which is still negligible in κ. This, however, does not make any meaningful improvement. To make a fair comparison, all our discussion in this work will focus on the same criteria of security error $2^{-\kappa}$.

Smaller GCs from Stronger Assumptions. There are also works focusing on constructing $o(|C|\kappa)$-bit garbled circuits. One line of works achieve fully succinct garbling, where the garbled circuit size is independent of $|C|$, based on heavy cryptographic primitives such as indistinguishability obfuscation (iO) [KLW15, BCG+18] or fully homomorphic encryption (FHE) plus attribute-based encryption (ABE) [GKP+13, BGG+14, HLL23]. These primitives are based on strong assumptions and incur a large computational overhead. More recently, another line of works construct $o(|C|\kappa)$-bit garbled circuits from homomorphic secret sharing (HSS) [MORS24, MORS25, ILL25], Puncturable Pseudorandom Function (PPRF) [CHHK25a, CHHK25b], or somewhat homomorphic encryption (SHE) [LWYY25b], providing better computational efficiency from broader assumptions (see Section 1.2 for a more detailed discussion).

This leads to our following question.

"Is $\Omega(|C|\kappa)$ inherent for garbled circuit size in the random oracle model with $2^{-\kappa}$ security error?"

1.1 Our Contribution

In this work, we answer the above question negatively by presenting the first garbling scheme in the random oracle model with garbled circuit size $o(|C|\kappa)$ and statistical security error $2^{-\kappa}$. Formally, we obtain the following result.

Theorem 1 *Assuming a (programmable) random oracle, there is a garbling scheme for any Boolean circuit C with garbled circuit size $O(|C| \log T + D\kappa^2 \log T)$ bits and input label size $O(W_I \log T)$ bits against any computationally unbounded adversary with T random oracle queries, where $|C|$ is the circuit size, D is the*

[1] For example in [RR21], the authors claimed in the introduction that exclusively using extremely efficient symmetric-key operations, the garbled circuit size must be at least $\Omega(\kappa)$ bits per gate.

circuit depth, W_I is the number of input wires, and κ is the statistical security parameter. The garbling scheme achieves statistical security with error $2^{-\kappa}$.

Our results are particularly strong when the adversary's runtime is much smaller than the (inverse of) desired statistical error. Taking the runtime T to be polynomial in κ, while still requiring inverse exponential statistical error $2^{-\kappa}$, we get the following corollary.

Corollary 1 *Assuming a random oracle, for any function $f(\kappa) = \omega(\log \kappa)$, there is a garbling scheme for any Boolean circuit C with garbled circuit size $O(|C|f(\kappa) + D\kappa^2 f(\kappa))$ bits and input label size $O(W_I f(\kappa))$ bits against any probabilistic polynomial-time adversary, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the statistical security parameter. The garbling scheme achieves statistical security with error $2^{-\kappa}$ for sufficiently large κ.*

In this setting, the achieved garbled circuit size is $o(|C|\kappa)$ bits when $D = o(|C|/\kappa f(\kappa))$. This even beats the recent work [CHHK25b] that achieves $O(|C|\kappa/\sqrt{\log \kappa})$ bits in the generic group model, which is much stronger than the random oracle model.

To better understand our results, we provide more details about the security model we studied and the comparison with previous works.

Security Model: In this work, we focus on the random oracle model, where the random oracle is modeled as an ideal functionality that computes a truly random function. The random oracle is assumed to be programmable (instead of a global random oracle [CV22]), whose output is programmed by the simulator in the ideal world. Every party can make queries to the random oracle and learn the corresponding function outputs. In this model, an adversary is computationally unbounded but can only make a bounded number of queries to the random oracle. We use T to denote the upper bound of the number of queries an adversary can make. This is the same setting as considered in most of the works on the lower bounds of garbled circuit size [ZRE15, BK24, FLZ24, XHX25, JRR25].

In this model, it is possible to achieve *statistical security*[2]. As discussed above, to make a fair comparison, we focus on statistical security with error $2^{-\kappa}$.

GC Size: The achieved garbled circuit size is $O(|C| \log T + D\kappa^2 \log T)$ bits with statistical security error $2^{-\kappa}$. In comparison, the best-known result [RR21] requires $O(|C|\kappa)$ bits to achieve statistical security with error $T \cdot 2^{-\kappa}$, which translates to $O(|C|(\kappa + \log T))$ bits with statistical security error $2^{-\kappa}$. Although our garbled circuit has the same asymptotic size in the adversary's computational power, we achieve a better asymptotic garbled circuit size in the statistical security parameter κ when $D = o(\sqrt{|C|})$.

We would like to highlight that our result is a theoretical result that only focuses on the *asymptotic* garbled circuit size in the security parameter κ. We leave the improvement of the concrete garbled circuit size to future work.

Input Label Size: The amortized label size per input bit in our construction is $O(\log T)$ bits. In contrast, all the mentioned works for Boolean circuits require $\Omega(\log T + \kappa)$ bits per input bit, including those using stronger assumptions [LWYY25b, MORS25, CHHK25b, ILL25].

Unlike previous works, our garbling scheme is not *projective*, i.e., we do not encode the input bit by bit. Instead, $O(\kappa)$ input bits are encoded together. Although the non-projectivity may lead to additional overhead when constructing secure 2PC protocols, we show that our construction does not suffer from this limitation.

Application to Constant-Round MPC: The most representative application of garbled circuits is to construct constant-round MPC protocols. In the context of n-party computation, most of the efficient constant-round protocols are constructed in the BMR template [BMR90, DI05], which computes a garbled circuit via a generic non-constant-round MPC (such as the GMW protocol). To achieve constant round

[2]However, when the random oracle is instantiated by a cryptographic hash function via Fiat-Shamir heuristic, the resulting protocol becomes computationally secure.

complexity, the framework [BMR90, DI05] requires that all wire labels can be computed within constant rounds of calls to the random oracle so that the garbling of each gate can be done in parallel.

Although this is indeed the case for Yao's garbled circuit [Yao86] and the Free-XOR variant [KS08], it is not for those [NPS99, ZRE15, RR21] with the row reduction optimization. This is because the row reduction optimization introduces dependency on wire labels, and the garbling algorithm has to compute the wire labels layer by layer. As a result, applying the BMR template [BMR90, DI05] on these garbling schemes does not yield a constant-round MPC protocol. The same limitation also appears in recent HSS-based, PPRF-based, and SHE-based garbling schemes [MORS24, LWYY25b, CHHK25a, CHHK25b, MORS25, ILL25].

On the other hand, our garbling scheme (with proper instantiation) does not introduce wire label dependency and can potentially be used to construct constant-round MPC protocols following the BMR template. We provide a detailed comparison of the garbling schemes in the random oracle model in Table 1.

Garbling scheme	Amortized GC size per gate AND	Amortized GC size per gate XOR	Wire label dependency
Textbook Yao [Yao86]	$8(\kappa + \log T)$	$8(\kappa + \log T)$	No
Point-Permute [BMR90]	$4(\kappa + \log T)$	$4(\kappa + \log T)$	No
Free-XOR [KS08]	$4(\kappa + \log T)$	0	No
Half-Gates [ZRE15]	$2(\kappa + \log T)$	0	Yes
Three-Halves [RR21]	$1.5(\kappa + \log T)$	0	Yes
Ours	$O(\log T)$	$O(\log T)$	No

Table 1: Comparison of garbling schemes in the random oracle model. We compare the GC sizes of all these garbling schemes for achieving a statistical error of $2^{-\kappa}$. Unlike the previous works, the circuit is not garbled gate by gate using our garbling scheme, so we only compare the amortized GC size per gate.

Main Results on Secure 2PC Protocol. We also construct a maliciously secure 2PC protocol based on our garbling scheme. The communication complexity of our construction is similar to the garbled circuit size, and the number of OTs is independent of $|C|$. The protocol only requires 1 OT round and 3 one-way communication rounds.

Theorem 2 *Assuming parallel oblivious transfers and a random oracle, there is a maliciously secure two-party computation protocol for any Boolean circuit C with communication of $O(|C| \log T + D(\log T + \kappa)^2 \log T + \mathsf{poly}(\kappa, \log T))$ bits plus $O(W_I + \mathsf{poly}(\kappa, \log T))$ invocations of OTs with message length $O(\log T)$ bits against any computationally unbounded adversary with T random oracle queries, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the statistical security parameter. The protocol achieves statistical security with error $2^{-\kappa}$ and only requires 1 OT round and 3 one-way communication rounds.*

Similarly, if taking the runtime T to be polynomial in κ, while still requiring inverse exponential statistical error $2^{-\kappa}$, we get the following corollary.

Corollary 2 *Assuming parallel oblivious transfers and a random oracle, for any function $f(\kappa) = \omega(\log \kappa)$, there is a maliciously secure two-party computation protocol for any Boolean circuit C with communication of $O(|C| f(\kappa) + D\kappa^2 f(\kappa) + \mathsf{poly}(\kappa))$ bits plus $O(W_I + \mathsf{poly}(\kappa))$ invocations of OTs with message length $O(f(\kappa))$ bits against any probabilistic polynomial-time adversaries, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the statistical security parameter. The protocol achieves statistical security with error $2^{-\kappa}$ for sufficiently large κ and only requires 1 OT round and 3 one-way communication rounds.*

In the literature of constant-round maliciously secure 2PC protocols,

- the best known result [HIV17] in the random oracle model achieves communication of $O(|C|\kappa)$[3] bits plus $O(|C|)$ OTs with message length κ bits only from black-box use of one-way functions;
- results with $o(|C|\kappa)$ bits of communication rely on somewhat homomorphic encryption [LWYY25a] or even stronger assumptions [QWW18, MPP20, HLL23].

In comparison, our result achieves $o(|C|\kappa)$ bits of communication with $O(W_I)$ OT calls. We discuss more related works on constant-round maliciously secure 2PC protocols based on symmetric cryptographic tools in Section 1.2.

Moreover, when we assume random bit-OT correlations and only the evaluator has outputs, we obtain a non-interactive secure computation (NISC) protocol with a similar communication cost.

Theorem 3 *Assuming $O(W_I \cdot \log T)$ random bit-OT correlations and a random oracle, there is a maliciously secure non-interactive secure computation (NISC) protocol for any Boolean circuit C with communication of $O(|C| \log T + D(\log T + \kappa)^2 \log T + \mathsf{poly}(\kappa, \log T))$ bits against any computationally unbounded adversary with T random oracle queries, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the statistical security parameter. The protocol achieves statistical security with error $2^{-\kappa}$.*

Taking T to be any polynomial in κ, our NISC protocol achieves a communication cost of $O(|C|f(\kappa) + D\kappa^2 f(\kappa) + \mathsf{poly}(\kappa))$ for any function $f(\kappa) = \omega(\log \kappa)$. Compared to a concurrent work [IJPS26] in the same setting that achieves $O(|C|\kappa)$ communication complexity, our NISC protocol achieves a better per-gate amortized communication cost, at the cost of an additional linear term in the circuit depth.

In addition, when we consider the computational security against PPT adversaries with advantage $2^{-\kappa}$, we show that the NISC protocol can also be constructed from a semi-malicious 2-round OT protocol in the common reference string (CRS) model with adaptive input choices. We provide the result here and refer the readers to K for more details.

Theorem 4 *For any function $f(\kappa) = \omega(\log \kappa)$, assuming a semi-malicious 2-round OT protocol (with message length $O(f(\kappa))$) with adaptive input choices in the CRS model (that achieves the computational security with advantage $2^{-\kappa}$ with $\mathsf{CC}_{\mathsf{OT}}(\kappa)$ communication cost) and a random oracle, there is a maliciously secure non-interactive secure computation protocol for any Boolean circuit C with communication of $O(|C|f(\kappa) + D\kappa^2 f(\kappa) + \mathsf{poly}(\kappa) + W_I \mathsf{CC}_{\mathsf{OT}}(O(\kappa)))$ bits against any probabllistic polynomial-time adversary, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the computational security parameter. The protocol achieves computational security with advantage $2^{-\kappa}$.*

1.2 Related Works

We introduce the related works in this section, where we focus on the asymptotic complexity of garbled circuit size and communication cost in the security parameter κ.

Lower Bounds on GC Size in the ROM. In the well-known Half-Gate garbling scheme [ZRE15], the authors present a garbling model called *linear garbling*, where the garbling and evaluation algorithms treat blocks of bits as elements in a ring and manipulate them via only linear combinations. All existing garbling schemes in the random oracle model at that time were built in this model. In this model, a 2κ-bit lower bound per AND gate is proved. Later on, in [RR21], the authors introduce the *slicing and dicing* technique, which is not in the linear garbling framework, and it achieves 1.5κ bits per AND gate. After that, [BK24, FLZ24] prove a lower bound of 1.5κ bits per AND gate from the slicing and dicing technique. The same lower bound is further proved in more generalized models in the recent two works [XHX25, JRR25]. However, all the above-mentioned lower bounds only focus on garbling a single AND gate, and the input wire labels are restricted to be completely random or random with respect to the XOR correlation. There is

[3] We only focus on the asymptotic communication cost in the security parameter κ here on the main term, i.e., the linear term in circuit size $|C|$.

also a work [LGW24] that proves a $2|C|\kappa$ lower bound when only PRFs are assumed, where the PRF keys are restricted to be the input labels.

Our garbling scheme deviates from the above models. In particular, each time a batch of $O(\kappa^2)$ gates in one layer are garbled together. This allows us to bypass the lower bounds mentioned above and achieve an amortized cost of $o(\kappa)$ bits per gate.

GCs from Broader Assumptions. Recently, garbled circuits (for Boolean circuits) with size of $o(|C|\kappa)$ bits have been constructed from various assumptions. The work [MORS25] achieves $O(|C|)$ bits based on homomorphic secret sharing from the Decisional Composite Residuosity (DCR) assumption. [CHHK25b] achieves $O(|C|\kappa/\sqrt{\log \kappa})$ bits in the generic group model, which can be weakened to the Power-DDH assumption if an additional $O(D \cdot \mathsf{poly}(\kappa))$ bits overhead is allowed. [LWYY25b] also achieves $O(|C|)$ bits assuming a variant of somewhat homomorphic encryption, which can be instantiated from the circular version of ring learning with errors (RLWE) or the Number Theory Research Unit (NTRU) assumption. Concurrently, [ILL25, LLL25] achieves $O(|C|)$ bits from homomorphic secret sharing based on either Power-DDH or Power-RLWE. These HSS-based, PPRF-based, and SHE-based garbled circuits [LWYY25b, CHHK25b, MORS25, ILL25] are all performed gate by gate, which makes them hard to support multiparty garbling within a constant number of rounds. Notably, a concurrent work [LLL25] yields linear garbled circuit size via HSS-based garbling of $\mathsf{poly}(\kappa)$ depth, which is the only known succinct garbling scheme whose garbling algorithm is not performed gate by gate.

Multiparty Garbling. Multiparty garbling protocols enable n parties to compute a garbled circuit jointly. A multiparty garbling protocol for RO-based garbled circuits can be computed in the BMR template using the techniques of [BMR90, DI05]. The resulting protocol supports a dishonest majority but increases the garbled circuit size by a factor of n (where n is the number of parties). However, as mentioned in Section 1.1, such a framework requires that all wire labels can be computed within constant rounds of calls to the random oracle so that the garbling of each gate can be done in parallel. For this reason, most of the constant-round MPC protocols have to use a garbling scheme without row reduction [Yao86, KS08] and the garbled circuit size is $4|C|n\kappa$ bits. The recent work [COPZ25] manages to save one ciphertext (and thus achieve $3|C|n\kappa$ bits of garbled circuit size) in the RO model with the half-gate optimization.

Another recent work [GLOS25] provides a different type of multiparty garbling protocols in the sub-optimal corruption setting where $t = (1-\epsilon)n$ for any constant $\epsilon \in (0,1)$. Their result can achieve $O(|C|\kappa + D\mathsf{poly}(\kappa))$ bits of garbled circuit size. Our garbling scheme is constructed from a multiparty garbling protocol of this type, and thus the work [GLOS25] directly provides a multiparty garbling protocol of our garbled circuit in a more efficient way compared to the BMR approach in the sub-optimal corruption setting.

Maliciously Secure 2PCs in the ROM. The most widely used approach of achieving constant-round maliciously secure 2PC protocols is the "cut-and-choose" technique [LP07], where $\Omega(\kappa)$ copies of the garbled circuits are generated, which results in $\Omega(|C|\kappa^2)$ bits of communication. Later, the work [NO09] presents the Large Efficient Garbled-circuit Optimization (LEGO) on the standard "cut-and-choose" technique, reducing the communication complexity to $O(|C|\kappa^2/\log|C|)$ bits. A line of follow-up works [FJN+13, FJNT15, NST17, KNR+17] then boost the concrete efficiency using LEGO.

Another line of works [WRK17, HSS17, HIV17, KRRW18, YWZ20, CWYY23, HKN+25, GLOS25] use the authenticated garbling approach to further improve the efficiency. In these works, only a single garbled circuit needs to be generated. In particular, [HIV17] achieves $O(|C|\kappa)$ bits of communication with $O(|C|)$ OTs with message length κ bits, which only has a constant overhead over the semi-honest protocol based on Yao's garbled circuits. However, the communication complexity of these works is restricted by the $\Omega(|C|\kappa)$ bits of garbled circuit size. Among them, the communication cost of [GLOS25] has the potential to be reduced using our improved garbled circuit. However, [GLOS25] requires 1 OT round plus 12 two-way communication rounds. In comparison, our construction only requires 1 OT round plus 3 one-way communication rounds.

2 Technical Overview

In this section, we give a high-level overview of our techniques. Our results assume a random oracle. We consider a fully malicious adversary who has unbounded computational power but can only access the random oracle T times. We aim to achieve statistical security with error $2^{-\kappa}$. In this setting, previous works can only achieve $\Omega(|C|(\kappa + \log T))$ bits of garbled circuit size using $(\kappa + \log T)$-bit long wire labels.

2.1 Starting Idea: Multiparty Garbling Protocol in the Head

Our approach is inspired by the recent advancements in multiparty garbling, a technique that allows n parties to collectively generate a garbled circuit securely against t corruptions in a constant number of rounds.

Our starting point is the recent work [GLOS25] which achieves $O(|C|\kappa + Dn^2\kappa)$ bits of both communication and garbled circuit size assuming a random oracle. At a very high level, their construction Π starts from a non-constant-round MPC protocol Π_0 that meets certain requirements. Then the output garbled circuit $\mathtt{GC}$ of Π consists of garbled circuits of each party's next-message functions in the underlying MPC protocol Π_0. For each party P_i, we use $\overrightarrow{\mathtt{GC}}_i$ to denote the garbled circuits of P_i's next-message functions. Thus $\mathtt{GC} = (\overrightarrow{\mathtt{GC}}_1, \overrightarrow{\mathtt{GC}}_2, \ldots, \overrightarrow{\mathtt{GC}}_n)$.

To convert it to a garbling scheme or a secure 2PC protocol, we simply let the garbler *emulate* the multiparty garbling protocol Π "in his head" and take the protocol output $\mathtt{GC} = (\overrightarrow{\mathtt{GC}}_1, \overrightarrow{\mathtt{GC}}_2, \ldots, \overrightarrow{\mathtt{GC}}_n)$ as the garbled circuit. In this way, the generated garbled circuit $\mathtt{GC}$ is "stronger" than an ordinary one, as the security remains even if the adversary learns the view of t parties in Π. Our constructions use this property in two different ways.

Achieving $o(|C|\kappa)$-bit GC. First, this property implies that the security remains even if the adversary can break the security of a bounded number ($\leq t$) of parties' garbled circuits for their next-message functions. Thus, to achieve statistical security with error $2^{-\kappa}$, we do not need each $\overrightarrow{\mathtt{GC}}_i$ to achieve statistical security with error $2^{-\kappa}$. Instead, it is sufficient to ensure that the adversary cannot break t of $\{\overrightarrow{\mathtt{GC}}_i\}_{i=1}^n$ *simultaneously* with probability $2^{-\kappa}$. In other words, we can weaken the security requirement of each $\overrightarrow{\mathtt{GC}}_i$ and use a smaller label size.

Jumping ahead, our construction only requires that each $\overrightarrow{\mathtt{GC}}_i$ cannot be broken with probability more than a small constant (1/40 to be precise) within T queries to the random oracle. When setting $t = \Theta(\kappa)$, this ensures that an adversary with T queries to the random oracle cannot break $t = \Theta(\kappa)$ of $\{\overrightarrow{\mathtt{GC}}_i\}_{i=1}^n$ with more than $2^{-\kappa}$ probability. This allows us to reduce the label size from $\kappa + \log T$ to $O(\log T)$.

Constructing Malicious 2PC with Constant Overhead. Second, when building a maliciously secure 2PC protocol, an evaluator can verify the correctness of $\mathtt{GC}$ by requesting the views of a random subset of t simulated parties. This ensures that the garbler computes most of $\{\overrightarrow{\mathtt{GC}}_i\}_{i=1}^n$ correctly. This feature allows us to achieve malicious security with essentially no additional communication overhead.

However, implementing the above two ideas requires additional care, as we will discuss below. In the following, we first give an overview of the multiparty garbling protocol in [GLOS25].

2.2 Review of GLOS Multiparty Garbling Protocol

The GLOS multiparty garbling protocol [GLOS25] is constructed by compiling a secret-sharing-based non-constant-round MPC protocol Π_0 via round collapsing. We will first illustrate this compilation process using the well-known DN protocol [DN07] as an example.

Compilation for DN. In the DN protocol, the execution begins with a preprocessing of random double-sharing pairs $([r]_t, [r]_{2t})$, where $[\cdot]_t$ denotes a Shamir sharing of degree t. We assume that the secrets and shares are binary for simplicity. Then, each party shares his input bit x as $[x]_t$. After these steps, the parties evaluate the target circuit gate by gate. Due to the linearity of Shamir secret sharing, all linear gates can

be evaluated locally without interaction. For a multiplication gate with inputs $[x]_t$ and $[y]_t$, the parties first compute locally $[z]_{2t} = [x]_t \cdot [y]_t + [r]_{2t}$. They then reconstruct the value z and derive $[x \cdot y]_t = z - [r]_t$. As a result, the evaluation phase only needs reconstructions of degree-$2t$ sharings, while all other work involves only local computation using the reconstructed secrets and the preprocessing/input data.

To obtain a constant-round MPC protocol, the idea in [GLOS25] is to let each party compute garbled circuits for his next-message functions in the evaluation phase and hope that the evaluator can simulate the entire DN execution. The main observation in [GLOS25] is that for each P_i, the input of his next-message functions only includes his shares in the preprocessing/input phase and the reconstruction results of degree-$2t$ sharings in the evaluation phase. When computing the garbled circuits for his next-message functions, P_i has received his shares in the preprocessing/input phase, and thus he may provide the associated input wire labels directly to the evaluator. The reconstruction results of degree-$2t$ sharings, however, are not known to P_i. To allow the evaluator to obtain the wire label associated with the reconstruction result of each degree-$2t$ Shamir sharing $[z]_{2t}$,

1. P_i will share his two wire labels k_0, k_1 by $[k_0]_{3t}, [k_1 - k_0]_t$ in the preprocessing phase.
2. With $[z]_{2t}$, all parties can locally compute $[k_z]_{3t} = [k_0]_{3t} + [z]_{2t} \cdot [k_1 - k_0]_t$. Each party P_j will modify the next-message function that computes his share of $[z]_{2t}$ to also compute his share of $[k_z]_{3t}$. Then P_j will garble the modified next-message functions.
3. In this way, the evaluator will obtain both $[z]_{2t}$ and $[k_z]_{3t}$ when simulating the DN execution. He can then reconstruct z and k_z, which allows him to evaluate P_i's next-round garbled circuit.

Towards $O(|C|\kappa)$ GC Size. In [GLOS25], each party simply uses Yao's garbled circuit to garble his (modified) next-message functions. The garbled circuit size $\mathtt{GC}$ is κ times of (1) the computation complexity of Π_0 plus (2) the communication complexity of Π_0. Since the computation and communication complexities of the DN protocol are $O(|C|n^2)$ as each $[z]_{2t}$ needs to be reconstructed to each party (as noted in [GLOS25], the king idea in [DN07] for reducing communication does not apply in their round collapsing technique), the resulting garbled circuit size is $O(|C|n^2\kappa)$ bits.

To further reduce the garbled circuit size, the authors of [GLOS25] use a packed-secret-sharing-based underlying protocol plus the network routing techniques [GPS21, GPS22] to construct an underlying protocol that only has $O(|C|)$ bits of communication and local AND operations when $t = n/4$, preserving the property that online communication only consists of public reconstructions of (packed) secret sharings. Then, with the freeXOR optimization of garbling [KS08], the garbled circuit size is reduced to $O(|C|\kappa)$ bits. We provide a more detailed overview about [GLOS25] in Section A.

2.3 Constructing Garbling Scheme with $o(|C|\kappa)$ GC size

Following our starting idea in Section 2.1, we can construct a garbling scheme where the garbling algorithm emulates the n-party garbling protocol in [GLOS25] and takes the protocol output $\mathtt{GC} = (\overrightarrow{\mathtt{GC}}_1, \overrightarrow{\mathtt{GC}}_2, \ldots, \overrightarrow{\mathtt{GC}}_n)$ as the garbled circuit. Recall that in [GLOS25], each party P_i's output $\overrightarrow{\mathtt{GC}}_i$ consists of a collection of garbled circuits of his next-message functions in the underlying non-constant-round MPC protocol Π_0. Also recall that each of such garbled circuits is just Yao's garbled circuit with the freeXOR optimization [KS08]. In particular, each party P_i will sample a random offset Δ. For each wire w, P_i will choose two random wire labels $k_{w,0}, k_{w,1}$ such that $k_{w,0} \oplus k_{w,1} = \Delta$. During the evaluation phase, the evaluator will learn exactly one of the two wire labels for each wire, and Δ remains hidden.

It has been proven in the literature that an adversary with T queries to the random oracle can break Yao's garbled circuit with freeXOR optimization with probability $T/2^{|\Delta|}$, where $|\Delta|$ is the length of Δ. Intuitively, in the random oracle model, the only attack an adversary can do is to guess Δ. Each guess can be verified by making a query to the random oracle. Since Δ is uniformly random, the success probability is bounded by $T/2^{|\Delta|}$. Thus, to achieve statistical security with error $2^{-\kappa}$, we have to set $|\Delta| = \kappa + \log T$.

We notice that the security of our garbling scheme remains even if the adversary can break t of $\{\overrightarrow{\mathtt{GC}}_i\}_{i=1}^n$. Thus, to achieve κ-bit security, it is sufficient to ensure that an adversary with T queries to the random oracle cannot break more than t of $\{\overrightarrow{\mathtt{GC}}_i\}_{i=1}^n$ *simultaneously* with probability more than $2^{-\kappa}$. For example:

- By choosing $|\Delta| = \log 5T$, the probability of breaking each $\overrightarrow{\mathtt{GC}}_i$ within T queries to the random oracle is bounded by $1/5$.
- Now we choose $t = \Theta(\kappa)$ and $n = 4t$ to instantiate the GLOS multiparty garbling protocol. The expected number of broken garbled circuits is bounded by $n/5 = 4t/5$.
- From the Chernoff bound, we may prove that with probability $1 - 2^{-\kappa}$, the adversary cannot break more than t parties' garbled circuits.

With the above chosen parameters, the garbled circuit size is $O(|C| \log T + D\kappa^2 \log T)$ bits.

Subtleties in the Security Proof. While the above idea is intuitive, proving the security of such a construction is not straightforward. Concretely, to show the security of the above garbling scheme, we need to construct a simulator that can simulate the view of the adversary.

When using κ bits for each wire label, the adversary cannot break *any* party's garbled circuits. Thus, it is sufficient to simulate the adversary's view by generating fake garbled circuits for each party. However, when using $\Theta(\log T)$ bits for each wire label, the adversary may break each party P_i's garbled circuits $\overrightarrow{\mathtt{GC}}_i$ with a noticeable probability (which is $1/5$ in the above concrete parameter choices). What makes it even worse is that when the simulator generates the garbled circuits, the simulator does not know which parties' garbled circuits will be attacked by the adversary. This restricts us from generating fake garbled circuits for each party directly, since an adversary may distinguish a correct garbled circuit from a fake one with noticeable probability due to such a smaller label size.

We address the above difficulty by relying on the property of the random oracle model.

- First, Yao's garbled circuit with freeXOR optimization only contains random binary strings. Thus, the simulator can generate a fake garbled circuit by just sampling random strings, whose distribution is identical to a real garbled circuit. In addition, when needed, the simulator can program the output of the random oracle to "explain" it as a real garbled circuit.
- Second, any attack made by the adversary (i.e., any guess of the offset Δ) has to be verified through queries to the random oracle. Thus, the simulator can detect which party's garbled circuits are under attack and whether the attack succeeds or not.

In summary, the simulator will sample random strings as garbled circuits of each party in the beginning. Then the simulator will keep track of the queries made by the adversary and detect whether each party P_i's garbled circuits are broken. If true, the simulator programs the output of the random oracle to explain P_i's simulated garbled circuits. If more than t parties' garbled circuits are broken, the simulator aborts, which only happens with probability $2^{-\kappa}$ under suitable choices of the parameters.

Remark 1 *We note that our security proof heavily relies on the random oracle model. The same idea seems hard to apply in the plain model. Indeed, after sending the garbled circuit to the evaluator (adversary), there is no way to "re-explain" each party's garbled circuits (like programming the output of the random oracle) or know which party's garbled circuits are broken (like monitoring the random oracle queries made by the adversary). It would be interesting to see whether a similar result can be achieved in the plain model.*

Summary of Our Garbling Scheme. We now summarize our garbling scheme. The garbling scheme contains a garbling algorithm Garble, an input encoding algorithm Encode, an evaluation algorithm Eval, and an output decoding algorithm Decode.

Garbling Algorithm. The garbling algorithm Garble takes the original circuit as its input and generates the garbled circuit together with the auxiliary information for input encoding.

1. The garbling algorithm first runs the preprocessing phase of the multiparty garbling protocol of [GLOS25]. This step generates the preprocessing data for the underlying non-constant-round protocol Π_0 together with the preprocessing for the compilation, i.e., the packed secret sharings for the labels of the secrets reconstructed in the online phase.

2. Then, the garbling phase of [GLOS25] is performed. The garbled circuits $\overrightarrow{\mathtt{GC}}_i$ of each party P_i's next-message functions are generated using freeXOR garbling [KS08] based on a random oracle, where each wire label is $O(\log T)$-bit long.

3. Finally, the output of Garble is computed:

   - The final garbled circuit $\mathtt{GC}$ contains $\{\overrightarrow{\mathtt{GC}}_i\}_{i=1}^n$ as well as the wire labels associated with each party's preprocessing data.
   - In the input phase of [GLOS25], parties share their inputs using packed secret sharings. Later, parties provide the evaluator with the wire labels associated with their input shares. Thus, the input encoding information in our garbling scheme contains the randomness for generating packed secret sharings for the inputs and the two wire labels for each bit of the sharings.

Input Encoding Algorithm. With the input encoding information, the encoding algorithm Encode can use the randomness to generate input packed sharings and get the corresponding wire labels.

Evaluation Algorithm. Given the garbled circuit $\mathtt{GC}$ obtained from Garble and the input labels obtained from Encode, we have all the input labels to evaluate $\{\overrightarrow{\mathtt{GC}}_i\}_{i=1}^n$, excluding the labels for the secrets reconstructed during the online phase of Π_0. Then, we can run the evaluation algorithm Eval. The evaluation algorithm runs the same evaluation process in [GLOS25]. More concretely:

1. The evaluation starts from evaluating the garbled circuits of all parties' first-round next-message functions that output the first packed secret sharing to be reconstructed in Π_0 together with a packed secret sharing of the associated wire labels. Note that the inputs of the first-round next-message functions only involve preprocessing data and input data, whose labels are known to the evaluation algorithm. Thus these garbled circuits can be evaluated.

2. After evaluating the garbled circuits in the first round, the evaluation algorithm gets the first packed secret sharing reconstructed in the online phase of Π_0 together with the packed secret sharing of the secrets' labels. The evaluator then reconstructs the secrets and the labels.

3. Now, for the garbled circuits in the second round, all input labels are obtained. The evaluator can then follow the first two steps to perform the evaluation. Eventually, after evaluating garbled circuits in each layer, the evaluation algorithm obtains the output of Π_0, which is set to be the output of Eval.

Output Decoding Algorithm. We assume that the underlying non-constant-round MPC protocol Π_0 will eventually output the function output directly. Thus, the decoding algorithm Decode simply outputs the output of Eval, which is already the function output.

Achieving Obliviousness and Authenticity. We can strengthen the basic construction to achieve obliviousness, where the function output is hidden given the output of Eval, and authentication, where a malicious evaluation algorithm cannot produce a fake output without being detected.

For obliviousness, we simply add a random mask on the output of Π_0, and the mask is provided as the output decoding information to Decode. To be more concrete, suppose $\boldsymbol{y}$ denotes the desired function output. We modify the functionality of Π_0 by taking an additional random string $\boldsymbol{r}$ as input and computing $\boldsymbol{r} + \boldsymbol{y}$. The garbling algorithm Garble further outputs $\boldsymbol{r}$ as the output decoding information. With $\boldsymbol{r}$, the output decoding algorithm Decode can remove the mask and obtain the desired function output.

For authenticity, a generic solution is to also prepare wire labels for each output bit and modify the evaluation algorithm Eval to also output the associated wire label for each output bit. These wire labels are

provided as the output decoding information to Decode, who checks the validity of the output from Eval. Here, the wire label serves as the MAC of each output bit. In this way, a malicious evaluation algorithm cannot generate a fake output that matches the wire labels.

In our case, however, the wire label is only $O(\log T)$-bit long, which is not sufficient to achieve statistical security with error $2^{-\kappa}$. While we may set the wire labels only for each output bit to be κ-bit long to achieve κ-bit security, this incurs a $\Theta(\kappa)$ multiplicative overhead depending on the output size. Our solution is to modify the functionality of Π_0 so that it outputs an encoding of the final outputs using a linear code with distance κ. In this way, a malicious evaluation algorithm has to guess κ labels to generate a fake output that is still a valid codeword. This allows us to achieve κ-bit security while still using $O(\log T)$-bit wire labels for the output.

Remark 2 *In our construction, the inputs can be arbitrarily separated into disjoint parts, each containing at most $O(\kappa)$ bits. The encoding algorithm can encode each part individually, and the output size is $O(\kappa \log T)$ bits independent of the size of each part. This property inherits from the multiparty garbling protocol in [GLOS25], where each party first shares his own inputs via packed secret sharing, and the evaluator needs to get the input label associated with each bit in the secret sharing. Thus, although our garbling scheme is not projective (which requires each input label to be dependent on a single input bit), it can achieve projectivity by splitting the inputs bit by bit and encoding each bit separately. However, the input label size per bit becomes $O(\kappa \log T)$ bits, and we lose the succinctness for the input label size.*

2.4 Maliciously Secure 2PC Protocol

Now we will overview our techniques for the maliciously secure 2PC protocol. Recall that our garbling scheme is based on the multiparty garbling protocol in [GLOS25]. To distinguish the actual parties for the 2PC protocol and the parties in the underlying multiparty garbling protocol, we will use *garbler* and *evaluator* to refer to the actual parties in the 2PC protocol and use *virtual parties* to refer to the parties in the underlying multiparty garbling protocol.

From GC to Semi-Honest 2PC. Recall that our garbling scheme is not projective. When using our garbling scheme to construct a 2PC protocol, we cannot let the evaluator obtain the input labels directly via Oblivious Transfers (OTs).

However, in the semi-honest case, we can simply let the evaluator share his inputs by himself and call OT to get the wire label for each bit of the sharing. While this may seem problematic as we reveal a partial of the input encoding information to the evaluator, we note that in [GLOS25], corrupted parties generate packed secret sharings of their inputs by themselves as well. Thus, we can argue the security of our semi-honest 2PC protocol following [GLOS25]. More concretely,

1. The garbler first runs the garbling algorithm and generates the garbled circuit. Then, the obtained encoding information contains all the labels for the wires whose values are from the input sharings. In particular, the garbler encodes his own inputs by himself and sends the two labels for each bit of the evaluator's input sharings to an OT functionality.

2. The evaluator generates the packed secret sharings for his inputs by himself. Then, the evaluator sends each bit of these sharings to the corresponding OT functionality to obtain its label.

3. The garbler then sends the garbled circuit together with his own input labels to the evaluator. Now, the evaluator is able to run the evaluation algorithm. The garbler also sends the output decoding information (if any) for the evaluator's outputs, so that the evaluator can compute his own outputs.

4. Finally, the evaluator sends the garbler's garbled output to him. The garbler holds the decoding information for his own outputs, and he can locally compute his outputs.

In this way, we obtain a semi-honest 2PC protocol whose communication complexity is the size of the garbled circuit plus the size of the input wire labels.

Security Against a Corrupted Evaluator. Now we will introduce how we generalize the protocol to achieve malicious security with abort. We first handle the security against a corrupted evaluator. For a corrupted evaluator, our fear is the following two points.

- First, the evaluator may use invalid packed secret sharings to encode his inputs.
- Second, the evaluator may return incorrect garbled outputs to the garbler.

We note that the second attack can be addressed by the authentication property of our garbling scheme, which prevents the evaluator from producing a fake output without being detected. To prevent the first attack, a generic solution is to let the garbler generate another *projective* garbled circuit for the input encoding algorithm Eval. However, the label size of the garbled circuit for Eval needs to be $\kappa + \log T$ bits to achieve κ-bit statistical security. As a result, this would incur a $\Theta(\kappa)$ multiplicative overhead depending on the input size of the evaluator.

To avoid this overhead, our solution continues to let the evaluator generate the packed secret sharings for his inputs but requires him to prove the validity of his sharings. In more detail, after the first OT round, the garbler asks the evaluator to open a random linear combination of all his packed secret sharings. To protect the privacy of the evaluator's inputs, the evaluator will input a random packed secret sharing as a mask. Now using the freeXOR technique, the evaluator can compute labels for the opened random packed secret sharing, which can be verified by the garbler. Then the garbler asks the evaluator to provide a small fraction of labels associated with the opened random packed secret sharing and verifies the correctness of those labels. This ensures that the evaluator uses correct shares for most of the virtual parties in the underlying multiparty garbling protocol. For the rest of the virtual parties, we simply think the evaluator has broken those virtual parties' garbled circuits.

Here, the readers may wonder why we cannot let the evaluator simply open all labels for the opened random packed secret sharing since the garbler can anyway compute those labels by himself. The reason is that this is true *only if* the garbler provides correct labels in the first OT round. If the garbler is malicious, he may not use wire labels satisfying the freeXOR requirements. Then, revealing all labels may leak the evaluator's input. In our check, we restrict the number of opened labels to be bounded by the privacy threshold of the underlying packed secret sharing.

Security Against a Corrupted Garbler. Now we focus on the security against a corrupted garbler. As we have pointed out in Section 2.1, the security of the underlying garbling scheme remains even if the evaluator learns $\Theta(t)$ virtual parties' views. We will utilize this property to detect the malicious behaviors of the garbler. When the garbler is malicious, our fear is the following three points.

- First, the garbler may use incorrect preprocessing data.
- Second, the garbler may compute incorrect garbled circuits for some virtual parties in the underlying multiparty garbling protocol.
- Finally, the garbler may perform a selective failure attack on OTs.

We note that in our garbling scheme, the preprocessing data only involves packed secret sharings. Thus, to prevent the first attack, we ask the garbler to prove the validity of all packed secret sharings generated in the preprocessing phase. Here, we use a similar approach to the verification of the evaluator's input packed secret sharings.

- The garbler first opens a random linear combination of all packed secret sharings.
- Then the evaluator asks the garbler to open a small random subset of virtual parties' views and verifies whether their views match the shares broadcast by the garbler.

In the meantime, the evaluator also checks whether these virtual parties' garbled circuits are correctly computed using their views. This allows the evaluator to detect the second attack as well.

However, the above verification is only useful if a malicious garbler cheats for many virtual parties. If only one virtual party's preprocessing data or garbled circuits are incorrect, with noticeable probability, his view will not be checked. As a result, such malicious behavior will not be caught by the evaluator. At a first glance, one may try to argue the security following [GLOS25]: we just view this virtual party as corrupted in the underlying multiparty garbling protocol [GLOS25], which can tolerate up to t corrupted parties. Unfortunately, our case is more subtle than [GLOS25] since all preprocessing data are generated and known by the garbler, whereas in [GLOS25], the adversary only learns the preprocessing data held by corrupted parties.

To further demonstrate the problem, consider the following selective failure attack launched by the garbler. Suppose the evaluator only has one input bit. Then the garbler may compute two possible evaluations of his generated garbled circuits, one using 0 as the evaluator's input, and the other one using 1, and obtain the wire labels in both cases. Now, suppose for some wire w in P_1's next-message function, the values in the two cases are different, say $w = 0$ if the evaluator's input is 0, and $w = 1$ otherwise. The garbler uses an incorrect wire label associated with $w = 1$ for this wire but computes the rest of the garbled circuits correctly. Now, during the evaluation, if the evaluator's input is 1, he will eventually obtain an incorrect output, which can be detected by the garbler, thus leaking his input to the garbler. What makes it even worse is that even if the evaluator notices that P_1's wire label associated with $w = 1$ is incorrect, he cannot abort since he would not notice this if his input is 0. As a result, the garbler would know that the evaluator's input is 1 if the evaluator aborts.

Thus, we need to ensure that if the above verification passes, an honest evaluator can always compute the correct output even if a small fraction of parties' preprocessing data or garbled circuits are incorrect. We notice that during the evaluation phase, besides evaluating each virtual party's garbled circuits, the evaluator only needs to reconstruct the secrets of packed secret sharings. To avoid the above attack, our construction uses a packed secret sharing scheme with error correction. In this way, even if a small fraction of parties' shares are incorrect, the evaluator can still recover the correct secrets.

Finally, we consider the possibility of a selective failure attack on the OTs. Concretely, suppose the OT messages corresponding to one bit of the evaluator's input packed secret sharing are (K_0, K_1). A malicious garbler may construct the garbled circuit using the correct labels K_0 and K_1, but send (K_0, K_1') in the OT, where $K_1' \neq K_1$. If the evaluator's choice bit is 0, the evaluator receives K_0, and the protocol proceeds normally. However, if the choice bit is 1, the evaluator receives the incorrect label K_1'. If the label is opened to the evaluator for verification, he will detect that the input labels are inconsistent and abort the protocol. Such an abort leaks this bit of the evaluator's input sharing to the garbler.

Fortunately, this attack can be mitigated by setting $t = \Omega(\kappa)$ in the multiparty garbling protocol. Under this choice, every block of κ bits in each input packed secret sharing is uniformly random. Consequently, in order to learn any information about the evaluator's input, the garbler would need to mount selective failure attacks on more than κ such OTs. However, this would cause the evaluator to abort with probability at least $1 - 2^{-\kappa}$. Hence, such a selective failure attack does not pose a significant security concern.

Reducing Rounds via Fiat-Shamir. In the above construction, after invoking the OT functionality to send the input labels to the evaluator, the two parties need 4 additional rounds for checking the evaluator's input sharings, another 4 rounds for checking the garbler's computation, and a final round for sending the garbler's outputs. In total, it requires 1 OT round plus 9 communication rounds in total. With random oracles, we can use the Fiat-Shamir transformation [FS86] to reduce the number of rounds. Concretely, during the check for the evaluator's input sharings, the proof can be compressed to a single message from the evaluator to the garbler with all challenges generated from the random oracle non-interactively. The case is similar for the check for the garbler's computation. As a result, we can reduce the interaction to 1 OT round and 3 one-way communication rounds. We refer the readers to Section 6 for our construction.

3 Preliminaries

Notation. Let $\mathbb{F}_q$ denote the finite field with q elements. We use $\boldsymbol{u} * \boldsymbol{v}$ to denote the coordinate-wise multiplication of two vectors $\boldsymbol{u}, \boldsymbol{v}$ of the same length, and we use $\boldsymbol{u} \otimes \boldsymbol{v}$ to denote the tensor product of two vectors. We use $A \equiv B$ to denote that two distributions A, B are the same.

3.1 Security Model

Our work studies the random oracle model. The random oracle is modeled as an ideal functionality that computes a truly random function whose output length is fixed. In short, we study the RO-hybrid model. The random oracle is assumed to be programmable, whose output is programmed by the simulator in the ideal world. Let κ denote the statistical security parameter.

We consider the security of our garbling scheme and MPC protocols against a computationally unbounded adversary who has at most T queries to the random oracle. We say two distributions are indistinguishable with statistical error $\epsilon(\kappa)$ when any computationally unbounded adversary with T random oracle calls can only distinguish the two distributions with advantage $\epsilon(\kappa)$. We aim to achieve statistical security with error $\epsilon = 2^{-\kappa}$.

3.2 Parallel Oblivious Transfers

To construct a maliciously secure 2PC protocol, oblivious transfers (OTs) are necessarily required. In particular, our malicious 2PC protocol is constructed assuming an OT round where the OTs run in parallel. We provide the functionality of parallel OTs of message length ℓ_{OT}.

Functionality $\mathcal{F}_{\mathsf{OT}}$

The trusted party interacts with two parties P_1, P_2 with public values L, ℓ_{OT}.

1. For each $i = 1, \ldots, L$, the trusted party receives two messages $m_0^{(i)}, m_1^{(i)} \in \{0,1\}^{\ell_{\mathsf{OT}}}$ from P_1 and receives a bit $b^{(i)} \in \{0,1\}$ from P_2.
2. For each $i = 1, \ldots, L$, the trusted party sends $m_b^{(i)}$ to P_2.

Figure 1: Functionality for parellel oblivious transfers.

3.3 Linear Secret Sharing Schemes

We follow the definition of linear secret sharing schemes (LSSSs) in [GLOS25], where an (n, t, k, ℓ)-LSSS over $\mathbb{F}_q$ is a secret sharing scheme that shares a secret vector in $\mathbb{F}_q^k$ to n shares, where each share is in $\mathbb{F}_q^\ell$. It is guaranteed that the joint distribution of any t shares is independent of the secret, and the sharing/reconstruction algorithms are all $\mathbb{F}_q$-linear. We also follow the definition from [GLOS25] to define the multiplicative properties of an LSSS. We need an LSSS Σ with 3-multiplicative reconstruction, i.e., there exist LSSSs $\Sigma^{(2)}, \Sigma^{(3)}$ such that the share-wise tensor product of two Σ-sharings for secrets $\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}$ is a $\Sigma^{(2)}$-sharing for $\boldsymbol{s}^{(1)} * \boldsymbol{s}^{(2)}$, and similarly the share-wise tensor product of a Σ-sharing and a $\Sigma^{(2)}$-sharing is a $\Sigma^{(3)}$-sharing for the bit-wise product of their secrets. We give the formal definitions in Section B.

We use $[\cdot]$ to denote a sharing of a LSSS Σ and use $[\cdot]^{(2)}, [\cdot]^{(3)}$ to denote the associated $\Sigma^{(2)}, \Sigma^{(3)}$-sharings. Additionally, a vector of m Σ-sharings $([\boldsymbol{s}_1], \ldots, [\boldsymbol{s}_m])$ can naturally be regarded as an LSSS $\Sigma_{\times m}$ over $\mathbb{F}_{2^m}$. $\Sigma_{\times m}$ (resp. $\Sigma^{(2)}_{\times m}$) is called an m-fold interleaved secret sharing scheme of Σ [CCXY18], we denote it by $[\cdot]_{\times m}$. Similarly for $\Sigma^{(2)}, \Sigma^{(3)}$-sharings.

3.4 Garbling Schemes

We follow the definition of garbling schemes in [BHR12]. We consider garbling schemes with respect to the leakage function $\Phi_{\mathsf{circ}}(C) = C$, i.e., the circuit itself is public.

Definition 1 ([BHR12]). *A garbling scheme consists of four algorithms:*

- $(\tilde{C}, e, d) \leftarrow \mathsf{Garble}(1^\kappa, C)$.
- $X := \mathsf{Encode}(e, x)$. *(deterministic)*
- $Y := \mathsf{Eval}(F, X, C)$. *(deterministic)*
- $y := \mathsf{Decode}(d, Y, C)$. *(deterministic)*

such that the following conditions hold.

Correctness: *For any circuit C and input x, after sampling $(\tilde{C}, e, d) \leftarrow \mathsf{Garble}(1^\kappa, C)$, it holds with probability 1 that $C(x) = \mathsf{Decode}(d, \mathsf{Eval}(\mathsf{Encode}(e, x)))$.*

Privacy: *There exists a simulator $\mathsf{Sim}_{\mathsf{pri}}$ such that for any circuit C and input x the following distributions are indistinguishable against any computationally unbounded adversary $\mathcal{A}$ with at most T random oracle calls, where $\mathsf{Sim}_{\mathsf{pri}}$ emulates $\mathcal{O}$ to interact with $\mathcal{A}$ in the ideal world.*

$$\boxed{\begin{array}{c}\mathtt{REAL}_{\mathsf{pri}}(1^\kappa, C, x, \mathcal{O}): \\ (\tilde{C}, e, d) \leftarrow \mathsf{Garble}(1^\kappa, C) \\ X := \mathsf{Encode}(e, x) \\ \textit{return } (\tilde{C}, X, d)\end{array}} \qquad \boxed{\begin{array}{c}\mathtt{IDEAL}_{\mathsf{pri}}(1^\kappa, C, C(x), \mathsf{Sim}_{\mathsf{pri}}): \\ (\tilde{C}, X, d) \leftarrow \mathsf{Sim}_{\mathsf{pri}}(1^\kappa, C, C(x)) \\ \textit{return } (\tilde{C}, X, d)\end{array}}$$

Obliviousness: *There exists a simulator $\mathsf{Sim}_{\mathsf{obl}}$ such that for any circuit C and input x the following distributions are indistinguishable against any computationally unbounded adversary $\mathcal{A}$ with at most T random oracle calls, where $\mathsf{Sim}_{\mathsf{obl}}$ emulates $\mathcal{O}$ to interact with $\mathcal{A}$ in the ideal world.*

$$\boxed{\begin{array}{c}\mathtt{REAL}_{\mathsf{obl}}(1^\kappa, C, x, \mathcal{O}): \\ (\tilde{C}, e, d) \leftarrow \mathsf{Garble}(1^\kappa, C) \\ X := \mathsf{Encode}(e, x) \\ \textit{return } (\tilde{C}, X)\end{array}} \qquad \boxed{\begin{array}{c}\mathtt{IDEAL}_{\mathsf{obl}}(1^\kappa, C, \mathsf{Sim}_{\mathsf{obl}}): \\ (\tilde{C}, X) \leftarrow \mathsf{Sim}_{\mathsf{obl}}(1^\kappa, C) \\ \textit{return } (\tilde{C}, X)\end{array}}$$

Authenticity: *For any circuit C and input x, no computationally unbounded adversary $\mathcal{A}$ with at most T random oracle calls can make the following distribution output* `TRUE` *with more than $2^{-\kappa}$ probability.*

$$\boxed{\begin{array}{c}(\tilde{C}, e, d) \leftarrow \mathsf{Garble}(1^\kappa, C) \\ X := \mathsf{Encode}(e, x) \\ Y \leftarrow \mathcal{A}(\tilde{C}, X) \\ \textit{return } \mathsf{Decode}(d, Y) \notin \{C(x), \perp\}\end{array}}$$

The garbled circuit size of a garbling scheme for a circuit C is defined to be the maximum size of $\tilde{C} \leftarrow \mathsf{Garble}(1^\kappa, C)$, and the input label size is the maximum size of $X = \mathsf{Encode}(e, x)$ for $e \leftarrow \mathsf{Garble}(1^\kappa, C)$.

3.5 Chernoff Bound

Let $\mathbb{E}(X)$ denote the expectation of a random variable X. Below is the well-known Chernoff bound in probability theory.

Lemma 1 (Chernoff Bound). *Suppose $X_1, \ldots, X_m$ are independent random variables taking values in $\{0, 1\}$, and let $X = X_1 + \cdots + X_m$ be their sum, and $\mathbb{E}(X) = \mu$. Then for any $\delta > 0$ it holds that:*

$$P(X \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2 \mu}{2+\delta}}.$$

Also for any $0 < \delta < 1$ it holds that:

$$P(X \leq (1-\delta)\mu) \leq e^{-\frac{\delta^2 \mu}{2}}.$$

4 The Underlying Protocol

We consider an abstract underlying protocol Π_0 that is built upon an (n, t, k, ℓ)-LSSS Σ over $\mathbb{F}_2$ with 3-multiplicative reconstruction. Let the associated LSSSs be $\Sigma^{(2)}, \Sigma^{(3)}$. We use $[\cdot], [\cdot]^{(2)}, [\cdot]^{(3)}$ to denote Σ-sharings, $\Sigma^{(2)}$-sharings, and $\Sigma^{(3)}$-sharings respectively. The underlying protocol is similar to the abstract protocol required in [GLOS25] with several slight modifications.

We now specify the structure and requirements of Π_0 for our garbling scheme below.

- Π_0 runs among m clients $P_1, \ldots, P_m$ and n servers $S_1, \ldots, S_n$.
- Π_0 is constructed in the $\{\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^{P_1}, \ldots, \mathcal{F}_{\mathsf{input}}^{P_m}\}$-hybrid model. The parties don't have inputs to $\mathcal{F}_{\mathsf{prep}}$, and the only inputs to $\mathcal{F}_{\mathsf{input}}^{P_j}$ are P_j's circuit inputs. The functionalities $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^{P_1}, \ldots, \mathcal{F}_{\mathsf{input}}^{P_m}$ are not restricted to having certain forms of outputs.
- An evaluation phase is run after the functionalities are invoked. The evaluation phase only involves servers' local computation and reconstructions of $\Sigma^{(2)}$-sharings (by letting each server send his share to the receiver), the receiver reconstructs the secret of each $\Sigma^{(2)}$-sharing after receiving the shares from all parties. Each server's local computation only depends on his outputs from $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^{P_1}, \ldots, \mathcal{F}_{\mathsf{input}}^{P_m}$ and the secrets of $\Sigma^{(2)}$-sharings reconstructed to him in the evaluation phase.
- After the evaluation phase, the servers output a $\Sigma^{(2)}$-sharing for each batch of output wires to the client who holds the wires. Then, the client reconstructs the secret as his outputs for these wires.

We require that the protocol satisfy perfect semi-honest security even if all the reconstructed $\Sigma^{(2)}$-sharings are made public. We further require that the corrupted servers' views can be generated one by one, i.e., the protocol satisfies a weaker version of semi-honest adaptive security, where the adaptive adversary should choose the corrupted clients at the beginning, and he can only adaptively corrupt a set of $\leq t$ servers at the end of the protocol. The formal requirement on the security of Π_0 is provided in Section C.

Such a protocol can be constructed by slightly modifying the underlying protocol in [GLOS25]. For completeness, we present the protocol and the security proof in Section D. We give the result here.

Lemma 2 *When $n = 4t$, there exist constants c, ℓ such that there are an $(n, t, k = cn, \ell)$-LSSS Σ and an explicit protocol building upon it satisfying the required properties of Π_0 with communication complexity of $O(|C| + Dn^2 + mn)$ bits, $O(|C| + Dn^2 + mn)$ local AND operations, an $O(|C| + Dn^2 + mn)$-bit large $\mathcal{F}_{\mathsf{prep}}$ output, and an $O(W_I^{P_i})$-bit large $\mathcal{F}_{\mathsf{input}}^{P_i}$ output for each $i = 1, \ldots, m$. Here $|C|$ is the number of AND gates in C, D is the circuit depth, and $W_I^{P_i}$ is the number of input wires attached to P_i.*

The underlying protocol for our 2PC is similar to the underlying protocol for the garbling scheme, except that we need $n = 6t$ to ensure that a $\Sigma^{(3)}$-sharing can have error-correction for $\leq t$ errors. In particular, the secret sharing schemes are instantiated based on algebraic geometry codes, which can be regarded as a modification of the packed Shamir secret sharing scheme [FY92] that is shared via an algebraic curve rather than a polynomial. Similar to the packed Shamir secret scheme, the error correction can be performed using the Berlekamp-Welch algorithm [WB86]. For more details about the linear secret sharing scheme, we refer the readers to Section E. We denote the protocol as Π_0'. For our 2PC, an additional property on Π_0' will be utilized is that all the outputs of functionalities $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^{P_1}, \ldots, \mathcal{F}_{\mathsf{input}}^{P_m}$ are random $\Sigma, \Sigma^{(2)}$-sharings (a server may hold the secrets of a part of the $\Sigma^{(2)}$-sharings) and random $\Sigma^{(2)}$-sharings with all zero secrets. The protocol Π_0' still requires communication of $O(|C| + Dn^2)$ bits, $O(|C| + Dn^2)$ local AND operations, and an $O(|C| + Dn^2)$-bit large $\mathcal{F}_{\mathsf{prep}}$ output, and an $O(W_I^{P_i})$-bit large $\mathcal{F}_{\mathsf{input}}^{P_i}$ output for each $i = 1, \ldots, m$. We refer the readers to Section D for more details.

5 Main Garbling Scheme

In this section, we present our main garbling scheme.

5.1 The Garbling Algorithm

We first give the garbling algorithm. The garbling algorithm initializes the underlying protocol Π_0 with $n, t = \Theta(\kappa)$ and then simulates the multiparty garbling of [GLOS25] to perform round collapsing on Π_0.

Let $\mathcal{O}$ be a random oracle with a sufficiently long ($\geq \ell \log 40T$) output length. Let $\mathcal{O}_1$ be an oracle that outputs the first $\log 40T + 1$ bits of the output from $\mathcal{O}$, and let $\mathcal{O}_2$ be an oracle that outputs the first $\ell \log 40T$ bits of the output from $\mathcal{O}$. The garbling algorithm Garble is presented as follows:

Algorithm $\mathsf{Garble}(\cdot,\cdot)$

Input: The security parameter 1^κ and a circuit C.

The algorithm $\mathsf{Garble}(1^\kappa, C)$ runs the following:

1. **Initializing.** Let W_I, W_O be the numbers of input and output wires in C respectively. Then:
   (a) Sample a W_O-bit string mask.
   (b) For each batch of k output wires of C (the batch of wires is indexed i), sample a random $4\kappa\ell$-bit string $\boldsymbol{r}_{\mathsf{en}}^{(i)}$. Without loss of generality, we assume that W_O is a multiple of k (otherwise we only need to add $< k$ zeros after the original output). For a W_O-bit string $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{W_O/k})$ where each $\boldsymbol{y}_i \in \mathbb{F}_2^k$, we define
   $$\mathsf{En}(\boldsymbol{y}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)})) = (\Sigma.\mathsf{Sh}(\boldsymbol{y}_1, \boldsymbol{r}_{\mathsf{en}}^{(1)}), \ldots, \Sigma.\mathsf{Sh}(\boldsymbol{y}_{W_O/k}, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)})).$$
   Inversely, for a $W_O n\ell/k$-bit string $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{W_O/k})$ where each $\boldsymbol{z}_i \in \mathbb{F}_2^{n\ell}$, we define
   $$\mathsf{De}(\boldsymbol{z}) = (\Sigma.\mathsf{Rec}(\boldsymbol{z}_1), \ldots, \Sigma.\mathsf{Rec}(\boldsymbol{z}_{W_O/k})).$$
   (c) Let $\overline{C}$ be the circuit
   $$\overline{C}(x, \mathsf{mask} \| \boldsymbol{r}_{\mathsf{en}}^{(1)} \| \ldots \| \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}) = \mathsf{En}(C(x) \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)})).$$
   Let Π_0 be the underlying protocol Π_0 based on an $(n = 4\kappa, t = \kappa, k = \Theta(n), \ell = \Theta(1))$-LSSS Σ over $\mathbb{F}_2$ with 3-multiplicative reconstruction that runs among three clients P, P', P'' and n servers $S_1, \ldots, S_n$ to compute the circuit $\overline{C}$, where all the input wires of C are attached to P', all the auxiliary input wires of $\overline{C}$ (i.e., $\mathsf{mask} \| \boldsymbol{r}_{\mathsf{en}}^{(1)} \| \ldots \| \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$) are attached to P'', and all the output wires of $\overline{C}$ are attached to P.
2. **Generating Preprocessing Data.** Emulate $\mathcal{F}_{\mathsf{prep}}$ to generate the preprocessing data in Π_0.
3. **Generating Input Data.** Randomly sample a randomness r_{input} for $\mathcal{F}_{\mathsf{input}}^{P'}$. Emulate $\mathcal{F}_{\mathsf{input}}^{P''}$ to generate the input data in Π_0 corresponding to the auxiliary inputs to $\overline{C}$.
4. **Generating Output Labels.** Assume that the output wires of $\overline{C}$ are batched into n_o batches in Step 1.(b), indexed $j = 1, \ldots, n_o$. For $i = 1, \ldots, \mathsf{rec}$, let the receiver of the i-th reconstruction of $\Sigma^{(2)}$-sharing during the online phase of Π_0 be R_i, and let the sharing be reconstructed be $[\boldsymbol{s}_i]^{(2)}$. For each $j = 1, \ldots, n_o$, we let $[\boldsymbol{s}_{\mathsf{rec}+j}]^{(2)}$ be the output sharing for the j-th batch of output wires, and we let R_i be the client P, as P holds the j-th batch of output wires. Then:
   (a) Sample $2k$ random $\log 40T$-bit strings as $\boldsymbol{r}_{b,\beta} = (r_{b,\beta}^{(1)}, \ldots, r_{b,\beta}^{(\log 40T)})$ for $b = 0, 1$ and $\beta = 1, \ldots, k$. Then, associate $(\boldsymbol{r}_{0,\beta}, \boldsymbol{r}_{1,\beta})$ with the β-th bit of $\boldsymbol{s}_i$. Let $\boldsymbol{r}_{\boldsymbol{b}}^{(\alpha)} = (r_{b,1}^{(\alpha)}, \ldots, r_{b,k}^{(\alpha)})$ for each $b = 0, 1$ and $\alpha = 1, \ldots, \log 40T$.
   (b) For $\alpha = 1, \ldots, \log 40T$, randomly sample a $\Sigma^{(3)}$-sharing $[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}$ and a Σ-sharing $[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]$ based on $\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}, \boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)}$.
   (c) For each $j = 1, \ldots, n$, compute
   $$Y_{(i-1)\ell^2+a,0}^{S_j} = \left(([\boldsymbol{r}_{\boldsymbol{0}}^{(1)}]^{(3)})_{[a\ell+1,(a+1)\ell]}^{S_j}, \ldots, ([\boldsymbol{r}_{\boldsymbol{0}}^{(\log 40T)}]^{(3)})_{[a\ell+1,(a+1)\ell]}^{S_j}\right)$$
   and
   $$Y_{(i-1)\ell^2+a,1}^{S_j} = \big(([\boldsymbol{r}_{\boldsymbol{0}}^{(1)}]^{(3)})_{[a\ell+1,(a+1)\ell]}^{S_j} \oplus ([\boldsymbol{r}_{\boldsymbol{1}}^{(1)} - \boldsymbol{r}_{\boldsymbol{0}}^{(1)}])^{S_j}, \ldots, ([\boldsymbol{r}_{\boldsymbol{0}}^{(\log 40T)}]^{(3)})_{[a\ell+1,(a+1)\ell]}^{S_j} \oplus ([\boldsymbol{r}_{\boldsymbol{1}}^{(\log 40T)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\log 40T)}])^{S_j}\big),$$

where $([\boldsymbol{s}])^{S_j}$ denotes S_j's share of $[\boldsymbol{s}]$ and $([\boldsymbol{s}]^{(3)})^{S_j}_{[c_1,c_2]}$ denotes the vector of the $c_1, c_1+1, \ldots, c_2$-th bits of S_j's share of $[\boldsymbol{s}]^{(3)}$.

Let $\{\boldsymbol{r_0}^{(\alpha)}, \boldsymbol{r_1}^{(\alpha)}\}_{\alpha=1}^{\log 40T}$ for $i = \mathsf{rec}+1, \ldots, \mathsf{rec}+n_o$ form a vector d_{mac}.

5. **Garble the Local Circuits.** For each $j = 1, \ldots, n$, let $\mathtt{Circ}^{S_j}$ be the circuit of S_j's local computation during the evaluation phase of Π_0. Then:
   (a) Sample a random $\log 40T$-bit string $\Delta^{S_j} \neq 0$.
   (b) For each wire w in $\mathtt{Circ}^{S_j}$ that is not an output wire of an XOR gate or an output gate, sample a random masking bit $\lambda_w \in \{0,1\}$ and a random $\log 40T$-bit string $k_{w,0}$.
   (c) For each XOR gate in $\mathtt{Circ}^{S_j}$ with input wires a, b and output wire o, compute $k_{o,0} = k_{a,0} \oplus k_{b,0}$ and $\lambda_o = \lambda_a \oplus \lambda_b$ gate by gate from the first layer.
   (d) For each wire w in $\mathtt{Circ}^{S_j}$ that is not an output wire of an output gate, compute $k_{w,1} = k_{w,0} \oplus \Delta^{S_j}$.
   (e) For each AND gate g in $\mathtt{Circ}^{S_j}$ with input wire a, b and output wire o and for each $(v_a, v_b) \in \{(0,0),(0,1),(1,0),(1,1)\}$, let $\chi_g(v_a, v_b) = (v_a \oplus \lambda_a) \wedge (v_b \oplus \lambda_b) \oplus \lambda_c$. Query the random oracle $\mathcal{O}_1$ with $k_{a,v_a} \| \alpha \| k_{b,v_b} \| v_b \| g$ and then compute the following ciphertext:
   $$\mathsf{ct}^{(g)}_{v_a,v_b} = \mathcal{O}_1(k_{a,v_a} \| v_a \| k_{b,v_b} \| v_b \| g) \oplus k_{o,\chi_g(v_a,v_b)} \| \chi_g(v_a, v_b).$$
   (f) For each output gate in $\mathtt{Circ}^{S_j}$ indexed α with input wire w and for each $i = 0, 1$, query the random oracle $\mathcal{O}_2$ with $k_{w,i} \| i \| w$ and then compute the following ciphertext:
   $$\mathsf{ct}_{w,i} = \mathcal{O}_2(k_{w,i} \| i \| w) \oplus Y_{\alpha, i \oplus \lambda_w}.$$
   (g) Let $X^{(j)}_{i,0} = k_{w_i,\lambda_{w_i}} \| \lambda_{w_i}, X^{(j)}_{i,1} = k_{w_i,1\oplus\lambda_{w_i}} \| (1 \oplus \lambda_{w_i})$, where w_i is the i-th input wire of $\mathtt{Circ}^{S_j}$. Then:
      - Let the F_j be the list of all the ciphertexts.
      - All labels $X^{(j)}_{i,b}$ for an input wire of index i whose value $b = 0, 1$ is from $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}^{P''}_{\mathsf{input}}$ (including the index of output bit from $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}^{P''}_{\mathsf{input}}$ which serves as this input wire) form a list L_j.
      - All labels $X^{(j)}_{i,b}$ for an input wire of index i whose value $b = 0, 1$ is from $\mathcal{F}^{P'}_{\mathsf{input}}$ (including the index of output bit from $\mathcal{F}^{P'}_{\mathsf{input}}$ which serves as this input wire) form a vector I_j.
      - Let d_j be the list of λ_w for all input wires w of output gates in $\mathtt{Circ}^{S_j}$.
6. **Encrypting Input Labels.** For $i = 1, \ldots, \mathsf{rec}$, the η-th bit of $\boldsymbol{s}_i$ is used as an input wire w in $\mathtt{Circ}^{R_i}$, compute
   $$\mathsf{ct}^{(i)}_{w,0} = \mathcal{O}_1(\boldsymbol{r}_{0,\eta} \| 0 \| i \| \eta \| w) \oplus k_{w,\lambda_w} \| \lambda_w$$
   and
   $$\mathsf{ct}^{(i)}_{w,1} = \mathcal{O}_1(\boldsymbol{r}_{1,\eta} \| 1 \| i \| \eta \| w) \oplus k_{w,1\oplus\lambda_w} \| (1 \oplus \lambda_w),$$
   where $\boldsymbol{r}_{0,\eta}, \boldsymbol{r}_{1,\eta}$ are the random strings associated with $\boldsymbol{s}_i$. Let F_{ct} be the vector of all the ciphertexts generated in this step.
7. **Computing Output.** Let $\tilde{C} = ((F_1, L_1, d_1), \ldots, (F_n, L_n, d_n), F_{\mathsf{ct}})$. For each input wire of C with index i, let $e = (r_{\mathsf{input}}, I_1, \ldots, I_n)$. Let $d = (\mathsf{mask}, d_{\mathsf{mac}})$. Output $(\tilde{C}, e, d)$.

Figure 2: The garbling algorithm.

5.2 The Encoding Algorithm

The encoding of the input is given by the encoding information e. The algorithm first uses the randomness of the input functionality $\mathcal{F}^{P'}_{\mathsf{input}}$ (parsed from e) and the function inputs to compute the output of $\mathcal{F}^{P'}_{\mathsf{input}}$. Then, the algorithm outputs the wire labels corresponding to the outputs of $\mathcal{F}^{P'}_{\mathsf{input}}$.

We present the encoding algorithm Encode below.

Algorithm $\mathsf{Encode}(\cdot,\cdot)$

Input: The encoding information e and the input x.

The algorithm $\mathsf{Encode}(e,x)$ runs the following:

- If e does not match the output form of Garble, output $\perp$.
- Otherwise:
  1. Parse r_{input} from e and compute the outputs of $\mathcal{F}_{\mathsf{input}}^{P'}$ from r_{input} and x.
  2. For each output bit b of $\mathcal{F}_{\mathsf{input}}^{P'}$, if it serves as the i-th input bit to $\mathtt{Circ}^{S_j}$, parse the label $X_{i,b}^{(j)}$ from I_j. Output the vector of all these labels.

Figure 3: The encoding algorithm.

5.3 The Evaluation Algorithm

Now we present the evaluation algorithm Eval. This algorithm is similar to the evaluation process of [GLOS25] that simulates the execution of Π_0, which is given below.

Algorithm $\mathsf{Eval}(\cdot,\cdot,\cdot)$

Input: A garbled circuit $\tilde{C}$, the garbled input X, and the original circuit C.

The algorithm $\mathsf{Eval}(\tilde{C},X,C)$ runs the following:

- If $\tilde{C}$ does not match the output form of Garble, output $\perp$. Otherwise, parse κ,n from the structure of $\tilde{C}$. If X does not match the output form of Encode, output $\perp$.
- Otherwise:
  1. For each $j=1,\ldots,n$, separate $\mathtt{Circ}^{S_j}$ to $\mathtt{Circ}_1^{S_j},\ldots,\mathtt{Circ}_{\mathsf{rec}}^{S_j}$, where rec is the number of reconstructions in the online phase of Π_0 corresponding to C,κ. Each $\mathtt{Circ}_i^{S_j}$ outputs S_j's share of the i-th $\Sigma^{(2)}$-sharing reconstructed in the evaluation phase of Π_0.
  2. For each $j=1,\ldots,n$, parse L_j and all the labels for the input shares of S_j from $\tilde{C},X$. All the input labels for $\mathtt{Circ}_1^{S_j}$ have already been prepared. Then, evaluate $\mathtt{Circ}_1^{S_j}$ gate by gate:
     - If g is an XOR gate with input wire a,b and output wire o, compute $k_{o,v_o\oplus\lambda_o}\|v_o\oplus\lambda_o=(k_{a,v_a\oplus\lambda_a}\|(v_a\oplus\lambda_a))\oplus(k_{b,v_b\oplus\lambda_b}\|(v_b\oplus\lambda_b))$.
     - If g is an AND gate with input wire a,b and output wire o, query the random oracle with $k_{a,v_a\oplus\lambda_a}\|(v_a\oplus\lambda_a)\|k_{b,v_b\oplus\lambda_b}\|(v_b\oplus\lambda_b)\|g$ and then compute
     $$\begin{aligned}&k_{o,v_o\oplus\lambda_o}\|v_o\oplus\lambda_o\\=&\mathcal{O}_1(k_{a,v_a\oplus\lambda_a}\|(v_a\oplus\lambda_a)\|k_{b,v_b\oplus\lambda_b}\|(v_b\oplus\lambda_b)\|g)\oplus\mathsf{ct}^{(g)}_{v_a\oplus\lambda_a,v_b\oplus\lambda_b}.\end{aligned}$$
     - If g is an output gate with input wire w (with index m), query the random oracle with $k_{w,v_w\oplus\lambda_w}\|(v_w\oplus\lambda_w)\|w$ and compute
     $$Y^{S_j}_{m,v_w}=\mathcal{O}_2(k_{w,v_w\oplus\lambda_w}\|(v_w\oplus\lambda_w)\|w)\oplus\mathsf{ct}_{w,v_w\oplus\lambda_w}.$$
  3. Compute $[\boldsymbol{r}_{\boldsymbol{s}_1}^{(\alpha)}]^{(3)}$ for $\alpha=1,\ldots,\log 40T$ from the output labels computed in the last step. If the obtained sharings are not valid $\Sigma^{(3)}$-sharings, output $\perp$. Otherwise, reconstruct the secrets $\{\boldsymbol{r}_{\boldsymbol{s}_1}^{(\alpha)}\}_{\alpha=1}^{\log 40T}$.
  4. For each $j=1,\ldots,n$ and input wire w of output gates in $\mathtt{Circ}_1^{S_j}$, parse λ_w from $\tilde{C}$. Then, compute $v_w=(v_w\oplus\lambda_w)\oplus\lambda_w$. Compute the sharing $[\boldsymbol{s}_1]^{(2)}$ from all these output wire values. If the obtained sharing is not a valid $\Sigma^{(2)}$-sharing, output $\perp$. Otherwise, reconstruct the secret s_1.
  5. If the β-th bit $s_{1,\beta}$ of $\boldsymbol{s}_1$ is used as an input wire w in $\mathtt{Circ}^{R_1}$, compute
     $$k_{w,s_{1,\beta}\oplus\lambda_w}\|(s_{1,\beta}\oplus\lambda_w)=\mathsf{ct}^{(1)}_{w,s_{1,\beta}}\oplus\mathcal{O}_1(\boldsymbol{r}_{s_{1,\beta},\beta}\|s_{1,\beta}\|1\|\beta\|w).$$

6. Evaluate $\mathtt{Circ}_2^{S_j}, \ldots, \mathtt{Circ}_{\mathsf{rec}}^{S_j}$ successively in the same way as in Steps 2-5 for all $j = 1, \ldots, n$. Eventually, all secrets $\boldsymbol{s}_i$ and their corresponding $\{\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}\}_{\alpha=1}^{\log 40T}$ are obtained for $i = \mathsf{rec}+1, \ldots, \mathsf{rec}+n_o$.
7. Output $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)} \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$.

Figure 4: The evaluation algorithm.

5.4 The Decoding Algorithm

The decoding algorithm is given as follows, where the MACs are checked, and the secrets of the reconstructed $\Sigma^{(2)}$-sharings are input to the circuit for P's output phase if the check passes.

Algorithm $\mathsf{Decode}(\cdot,\cdot,\cdot)$

Input: The decoding information d, the garbled output Y, and the original circuit C.

The algorithm $\mathsf{Decode}(d, Y, C)$ runs the following:

- If d, Y do not match the output form of $\mathsf{Garble}, \mathsf{Eval}$, output $\perp$.
- Otherwise:
  1. For each batch of output wires indexed $j = 1, \ldots, n_o$ of $\overline{C}$, parse $\boldsymbol{s}_i$ and $\{\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}\}_{\alpha=1}^{\log 40T}$ from Y, and parse $\{\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}, \boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)}\}_{\alpha=1}^{\log 40T}$ from d. Check whether the sharings are all valid. If not, output $\perp$.
  2. Parse mask from d. Check whether $\boldsymbol{s}_i$ and $\{\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}\}_{\alpha=1}^{\log 40T}$ match $\{\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}, \boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)}\}_{\alpha=1}^{\log 40T}$ for each $i = \mathsf{rec}+1, \ldots, \mathsf{rec}+n_o$, i.e. whether
  $$\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)} = \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)} + \boldsymbol{s}_i * (\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)})$$
  for each $\alpha = 1, \ldots, \log 40T$. If not, output $\perp$. Otherwise, P's outputs $\overline{y}$ of $\overline{C}$ are obtained from $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$.
  3. Check whether $\overline{y}$ is a valid output of En (i.e., check whether each batch of outputs from $C(x)\|\mathsf{mask}$ is shared via a valid Σ-sharing). If not, output $\perp$. Otherwise, compute the circuit outputs of C by $\mathsf{De}(\overline{y}) \oplus \mathsf{mask}$.

Figure 5: The decoding algorithm.

We restate the main theorem for our garbling scheme here, which lists the parameters we achieve.

Theorem 1 *Assuming a (programmable) random oracle, there is a garbling scheme for any Boolean circuit C with garbled circuit size $O(|C| \log T + D\kappa^2 \log T)$ bits and input label size $O(W_I \log T)$ bits against any computationally unbounded adversary with T random oracle queries, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the statistical security parameter. The garbling scheme achieves statistical security with error $2^{-\kappa}$.*

We provide the security proof of our garbling scheme in Section F and give a detailed cost analysis in Section G.

6 Maliciously Secure 2PC

In this section, we give our maliciously secure 2PC protocol based on our garbling scheme. For simplicity, the 2PC protocol Π presented in this section achieves a statistical security against the adversary with statistical error $\Theta(1)/2^{\kappa}$. To further reduce the error to $1/2^{\kappa}$, we may simply replace κ in the protocol by $\kappa + \mathsf{const}$ for a large enough constant $\mathsf{const} = O(1)$ respectively. This will not affect the final communication complexity.

Let $\mathcal{O}$ be a random oracle with a sufficiently large (still $\mathsf{poly}(\kappa, \log T)$) output length. Let $\mathcal{O}_1$ be an oracle that outputs the first $\log 40T + 1$ bits of the output from $\mathcal{O}$, and let $\mathcal{O}_2$ be an oracle that outputs the first $\ell \log 40T$ bits of the output from $\mathcal{O}$. Our protocol Π runs 4 rounds $\Pi_1, \Pi_2, \Pi_3, \Pi_4$ in order. The messages in each round are sent together.

The First Round. The first round Π_1 is the OT round, where the garbler generates the garbled circuit and computes the wire labels for the evaluator's input wires. Then, the garbler and the evaluator invoke the OT functionality to let the evaluator get the input labels for his inputs. In addition, the garbler sends the first challenge for checking the evaluator's input sharings.

Protocol Π_1

The protocol is run by a garbler G and an evaluator E.

1. **Generating Output Mask.** Let W_O be the number of output wires attached to G. At the beginning of the protocol, the garbler samples a random mask $\mathsf{mask} \in \mathbb{F}_2^{W_O}$ for his output. For each batch of k output wires indexed i attached to G, G samples a random $n = (882\ell(\log T + \kappa) + 150\kappa)$-bit string $\boldsymbol{r}_{\mathsf{en}}^{(i)}$. We use the same algorithms En, De as in Garble. For the original circuit $C(x_1, y_1) = (C_G(x_1, y_1), C_E(x_1, y_1))$, where $C_G(x_1, y_1)$ (resp. $C_E(x_1, y_1)$) is the output for G (resp. E), we use $\overline{C}$ to denote the circuit:
$$\begin{aligned}&\overline{C}(x_G \| \mathsf{mask} \| \boldsymbol{r}_{\mathsf{en}}^{(1)} \| \ldots \| \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}, x_E)\\ =&(\mathsf{En}(C_G(x_G, x_E) \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)})), C_E(x_G, x_E)).\end{aligned}$$
2. **Initializing.** Let Π_0' be the underlying protocol based on an $(n = 6t, t = 147 \log T + 172\kappa, k, \ell)$-LSSS Σ over $\mathbb{F}_2$ that runs among two clients G, E and n servers $S_1, \ldots, S_n$ to compute the circuit $\overline{C}$, where G's input is $x_G \| \mathsf{mask} \| \boldsymbol{r}_{\mathsf{en}}^{(1)} \| \ldots \| \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$, and E's input is x_E. Here x_G, x_E are G, E's inputs to C, respectively.
3. **Generating Preprocessing and Input Data.** G emulates $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^G$ to generate the preprocessing data and his input sharings in Π_0'. E also emulates $\mathcal{F}_{\mathsf{input}}^E$ to generate his input sharings in Π_0'.
4. **Garbling the Circuit.** Assume that the output wires of $\overline{C}$ attached to G are batched into n_o batches, indexed $j = 1, \ldots, n_o$. For $i = 1, \ldots, \mathsf{rec}$, let the receiver of the i-th reconstruction of $\Sigma^{(2)}$-sharing during Π_0' online phase be R_i, and let the sharing to be reconstructed be $[\boldsymbol{s}_i]^{(2)}$. For each $j = 1, \ldots, n_o$, we let $[\boldsymbol{s}_{\mathsf{rec}+j}]^{(2)}$ to be the output sharing for the j-th batch of output wires attached to G, and we let the receiver $R_{\mathsf{rec}+j}$ to be the client G, as G holds the j-th batch of output wires. Then, G runs Steps 4(a)-4(c),5,6 of Garble and obtains the garbled circuit $\tilde{C}$ corresponding to Π_0'.
5. **Getting Input Labels.** Initializing $\mathcal{F}_{\mathsf{OT}}$ with L to be the total size of input sharings (for OT instances used in this step) plus $\lceil \kappa / \log 40T + 1 \rceil$ (used in Step 6) plus $\lceil n \cdot \ell \cdot (n + \kappa) / \log 40T \rceil$ (Step 7) plus $\lceil n^2 \cdot \ell_{\mathcal{O}} / \log 40T \rceil$ (Step 9), where $\ell_{\mathcal{O}}$ is the output length of $\mathcal{O}$. The message length ℓ_{OT} of $\mathcal{F}_{\mathsf{OT}}$ is set to be $\log 40T + 1$. For each output bit b of $\mathcal{F}_{\mathsf{input}}^E$, if it serves as the i-th input to $\mathtt{Circ}^{S_j}$, G sends $X_{i,0}^{(j)}, X_{i,1}^{(j)}$ to $\mathcal{F}_{\mathsf{OT}}$, and E sends b to $\mathcal{F}_{\mathsf{OT}}$. E then receives all the wire labels corresponding to the output of $\mathcal{F}_{\mathsf{input}}^E$.
6. **Sending Challenge.** G samples a random $(\kappa + \log T)$-bit string s and sends it to E (via $\lceil \kappa / \log 40T + 1 \rceil$ OT instances).
7. **Generating the Mask Sharing.** Let $\kappa' = n + \kappa$. E generates a random $\Sigma_{\times\kappa'}$-sharing $[\boldsymbol{r}_0]_{\times\kappa'}$. For each bit b (indexed i) of a server S_j's share of the sharing, G generates a label $L_{i,0} \in \mathbb{F}_2^{\log 40T}$ and computes $L_{i,1} = L_{i,0} \oplus \Delta^{S_j}$. Then, G sends them to $\mathcal{F}_{\mathsf{OT}}$, and E sends b to $\mathcal{F}_{\mathsf{OT}}$. Then, E receives $L_{i,b}$ from $\mathcal{F}_{\mathsf{OT}}$.
8. **Generating Auxiliary Sharings.** G generates a random $\Sigma_{\times\kappa'}$-sharing $[\boldsymbol{r}_1]_{\times\kappa'}$, a random $\Sigma^{(2)}_{\times\kappa'}$-sharing $[\boldsymbol{r}_2]^{(2)}_{\times\kappa'}$, a random $\Sigma^{(3)}_{\times\kappa'}$-sharing $[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$, and a random $\Sigma^{(2)}_{\times\kappa'}$-sharing $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ with an all-zero secret $\boldsymbol{o} = (0, 0, \ldots, 0) \in \mathbb{F}^k_{2^{\kappa'}}$.
9. **Committing to Servers' View.** For each $j \in \mathcal{V}$, let View^{S_j} be the set of:
   - Δ^{S_j};
   - $\lambda_w, k_{w,0}$ for all wires w excluding output wires of XOR/output gates in $\mathtt{Circ}^{S_j}$, all the outputs to S_j from $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^G$ excluding S_j's shares of the $\Sigma^{(2)}$-sharings whose secrets are held by other servers;
   - S_j's shares of all $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ corresponding to reconstructions of $\Sigma^{(2)}$-sharings whose receiver is not S_j;
   - all the sharings $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ corresponding to reconstructions of $\Sigma^{(2)}$-sharings whose receiver is S_j;
   - S_j's shares of $[\boldsymbol{r}_1]_{\times\kappa'}$, $[\boldsymbol{r}_2]^{(2)}_{\times\kappa'}$, $[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$, and $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$.

Then, G samples a random string $\boldsymbol{r}_{\mathsf{commit}}^{S_j} \in \mathbb{F}_2^{\kappa+\log T}$ and queries the random oracle to get $c_j = \mathcal{O}(\mathsf{View}^{S_j} \| \boldsymbol{r}_{\mathsf{commit}}^{S_j} \| j)$. Finally, G sends $(c_1, \ldots, c_n)$ to E (via $\lceil n \cdot \ell_{\mathcal{O}} / \log 40T \rceil$ OT instances).

For each pair of servers (S_i, S_j), let all the shares for S_j from $\mathcal{F}_{\mathsf{prep}}$ of the sharings whose secrets are held by S_i form a set $\mathsf{Sh}_{i,j}$. Then, G samples a random string $\boldsymbol{r}_{\mathsf{comsh}}^{(i,j)} \in \mathbb{F}_2^{\kappa+\log T}$ and queries the random oracle to get $\sigma_{i,j} = \mathcal{O}(\mathsf{Sh}_{i,j} \| \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)} \| i \| j)$. Finally, G sends $\sigma_{i,j}$ to E (via $\lceil n(n-1) \cdot \ell_{\mathcal{O}} / \log 40T \rceil$ OT instances for all pairs of (i, j)).

Figure 6: The first round of our maliciously secure 2PC.

The Second Round. In the second round Π_2, the evaluator responds to the challenge generated by the garbler by sending a linear combination of his input shares. The result is used as a seed for generating a random set of servers whose share labels are required to open to the garbler. The random set is generated following the Fiat-Shamir approach [FS86] via the random oracle.

Protocol Π_2

1. **Computing Coefficients.** Let the number of input Σ-sharings output by $\mathcal{F}_{\mathsf{input}}^E$ be k_0. E queries $\mathcal{O}$ with the challenge s and let the first $\kappa' \cdot k_0$ bits of $\mathcal{O}(s)$ be $\theta_1, \ldots, \theta_{k_0} \in \mathbb{F}_{2^{\kappa'}}^{k_0}$.
2. **Computing Linear Combination.** E regards his input sharings $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ as $\Sigma_{\times\kappa'}$-sharings $[\boldsymbol{I}_1]_{\times\kappa'}, \ldots, [\boldsymbol{I}_{k_0}]_{\times\kappa'}$ (by embedding each bit into $\mathbb{F}_{2^{\kappa'}}$). Then, E computes:
$$[\boldsymbol{v}]_{\times\kappa'} = \sum_{i=1}^{k_0} \theta_i [\boldsymbol{I}_i]_{\times\kappa'} + [\boldsymbol{r}_0]_{\times\kappa'}$$
and sends it to G.
3. **Generating the Challenge for Opening Shares.** E queries $\mathcal{O}$ with $[\boldsymbol{v}]_{\times\kappa'}$ and obtains a subset $\mathcal{U}$ of $t - \kappa$ servers among $S_1, \ldots, S_n$ from the first n bits of $\mathcal{O}([\boldsymbol{v}]_{\times\kappa'})$. E sends $\mathcal{U}$ to G.
4. **Opening Input Labels.** For each $S_j \in \mathcal{U}$, let the circuit computing S_j's share of
$$[\boldsymbol{v}]_{\times\kappa'} = \sum_{i=1}^{k_0} \theta_i [\boldsymbol{I}_i]_{\times\kappa'} + [\boldsymbol{r}_0]_{\times\kappa'}$$
from his shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ be $\mathtt{RanLin}_j$, which is a linear circuit only containing XOR gates. Note that E holds the input wire labels. E then computes the wire label for each wire in $\mathtt{RanLin}_j$ gate by gate. The output wire label for each XOR gate is set to be the XOR of two input wire labels to this gate. Finally, E obtains the output wire labels for S_j's share of $[\boldsymbol{v}]_{\times\kappa'}$ for all $S_j \in \mathcal{U}$ and sends them to G.
5. **Sending Challenge.** E samples a random $(\kappa + \log T)$-bit string s' and sends it to G.

Figure 7: The second round of our maliciously secure 2PC.

The Third Round. In the third round Π_3, the garbler first checks the input sharings of the evaluator. Then, the garbler responds to the challenge generated by the evaluator and generates the challenge set of servers for checking his garbling via Fiat-Shamir [FS86]. Then, the garbler computes the authentication information for his garbled circuit, i.e., the views of the challenge set of servers during the multiparty garbling, using the challenges. Finally, the garbler sends the garbled circuit together with the additional authentication information to the evaluator.

Protocol Π_3

1. **Checking Evaluator's Input Sharings.** G queries $\mathcal{O}$ with the challenge s to get $\theta_1, \ldots, \theta_{k_0} \in \mathbb{F}_{2^{\kappa'}}$, and G also queries $\mathcal{O}$ with $[\boldsymbol{v}]_{\times\kappa'}$. Then G verifies whether:

- The set $\mathcal{U}$ received from E matches $\mathcal{O}([\boldsymbol{v}]_{\times\kappa'})$.
- The output sharing $[\boldsymbol{v}]_{\times\kappa'}$ and the labels of each server $S_j \in \mathcal{U}$'s share matches the output wire labels of $\mathtt{RanLin}_j$ (computed from the wire labels he sent to $\mathcal{F}_{\mathsf{OT}}$).
- The sharing $[\boldsymbol{v}]_{\times\kappa'}$ is a valid $\Sigma_{\times\kappa'}$-sharing.

If not, G aborts the protocol.

2. **Computing Coefficients.** Recall that $\kappa' = n + \kappa$. Let the number of Σ-sharings (resp. $\Sigma^{(2)}$-sharings, $\Sigma^{(3)}$-sharings, $\Sigma^{(2)}$-sharings with all-zero secrets) generated by G while emulating $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}^{G}_{\mathsf{input}}$ and during Step 4 of Garble be k_1 (resp. k_2, k_3, k_4). G queries $\mathcal{O}$ with the challenge s' and let the first $\kappa'(k_1+k_2+k_3+k_4)$ bits of $\mathcal{O}(s')$ be $(\alpha_1, \ldots, \alpha_{k_1}, \beta_1, \ldots, \beta_{k_2}, \gamma_1, \ldots, \gamma_{k_3}, \eta_1, \ldots, \eta_{k_4})$ in $\mathbb{F}_{2^{\kappa'}}^{k_1+k_2+k_3+k_4}$.
3. **Generating Verification Sharings.** Let the k_1 Σ-sharings counted in the last step be $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, let the k_2 $\Sigma^{(2)}$-sharings be $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, let the k_3 $\Sigma^{(3)}$-sharings be $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, and let the k_4 $\Sigma^{(2)}$-sharings with all-zero secrets be $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$. G regards them as $\Sigma_{\times\kappa'}, \Sigma^{(2)}_{\times\kappa'}, \Sigma^{(3)}_{\times\kappa'}$-sharings (by embedding each bit into $\mathbb{F}_{2^{\kappa'}}$). Then, G computes VerSh as the set of

$$\Sigma_{i=1}^{k_1} \alpha_i [\boldsymbol{x}_i]_{\times\kappa'} + [\boldsymbol{r}_1]_{\times\kappa'}, \qquad \Sigma_{i=1}^{k_2} \beta_i [\boldsymbol{y}_i]^{(2)}_{\times\kappa'} + [\boldsymbol{r}_2]^{(2)}_{\times\kappa'},$$

$$\Sigma_{i=1}^{k_3} \gamma_i [\boldsymbol{z}_i]^{(3)}_{\times\kappa'} + [\boldsymbol{r}_3]^{(3)}_{\times\kappa'}, \text{ and } \Sigma_{i=1}^{k_4} \eta_i [\boldsymbol{o}_i]^{(2)}_{\times\kappa'} + [\boldsymbol{o}]^{(2)}_{\times\kappa'}.$$

4. **Generating the Challenge for Opening Views.** Let $\tilde{C} = ((F_1, L_1, d_1), \ldots, (F_n, L_n, d_n), F_{\mathsf{ct}})$, and let

$$\mathsf{GC} = (\tilde{C}, c_1, \ldots, c_n, \{\sigma_{i,j}, \sigma_{j,i}\}_{1 \le i < j \le n}).$$

Then, G queries $\mathcal{O}$ with $\mathsf{GC} \| \mathsf{VerSh}$ and lets the first n bits of the output denote a set $\mathcal{V}$ of $t/3$ indices in $\{1, \ldots, n\}$.

5. **Sending the Garbled Circuit.** G sends $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}^{S_j}_{\mathsf{commit}}\}_{j\in\mathcal{V}}, \{\mathsf{Sh}_{i,j}, \boldsymbol{r}^{(i,j)}_{\mathsf{comsh}}, \mathsf{Sh}_{j,i}, \boldsymbol{r}^{(j,i)}_{\mathsf{comsh}}\}_{i\in\mathcal{V}, j\in\{1,\ldots,n\}\setminus\{i\}})$ to E.

Figure 8: The third round of our maliciously secure 2PC.

The Fourth Round. In the fourth round Π_4, the evaluator verifies the authentication information of the garbled circuit. If the check passes, he can perform the evaluation on the garbled circuit. Finally, the evaluator obtains his own circuit outputs and sends the GC outputs to the garbler. The garbler then decodes his own outputs.

Protocol Π_4

1. **Verifying the Garbled Circuit.** The evaluator E checks whether the following hold:
   - The 4 sharings in VerSh are valid $\Sigma_{\times\kappa'}$, $\Sigma^{(2)}_{\times\kappa'}$, $\Sigma^{(3)}_{\times\kappa'}$-sharings and a $\Sigma^{(2)}_{\times\kappa'}$-sharing with an all-zero secret.
   - $c_j = \mathcal{O}(\mathsf{View}^{S_j} \| \boldsymbol{r}^{S_j}_{\mathsf{commit}} \| j)$ for each $j \in \mathcal{V}$, and $\sigma_{i,j} = \mathcal{O}(\mathsf{Sh}_{i,j} \| \boldsymbol{r}^{(i,j)}_{\mathsf{comsh}} \| i \| j)$, $\sigma_{j,i} = \mathcal{O}(\mathsf{Sh}_{j,i} \| \boldsymbol{r}^{(j,i)}_{\mathsf{comsh}} \| j \| i)$ for each $j \in \mathcal{V}, i \in \{1, \ldots, n\} \setminus \{j\}$.
   - For the output $\Sigma^{(2)}$-sharings from $\mathcal{F}_{\mathsf{prep}}$ to $j \in \mathcal{V}$ where S_j holds the secrets, the shares from $\{\mathsf{Sh}_{j,i}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$ form valid sharings, and the secret in View^{S_j} matches the sharings.
   - $\tilde{C}, d^{E}_{\mathsf{prep}}, \mathsf{VerSh}$, and the outputs received from $\mathcal{F}_{\mathsf{OT}}$ match $\{\mathsf{View}^{S_j}\}_{j\in\mathcal{V}}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i\in\mathcal{V}, j\in\{1,\ldots,n\}\setminus\{i\}}$. More concretely, the following elements in $\tilde{C}, d^{E}_{\mathsf{prep}}, \mathsf{VerSh}$, and outputs received from $\mathcal{F}_{\mathsf{OT}}$ can be computed from $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$:
     - (F_j, L_j, d_j) and the ciphertexts in F_{ct} generated in Step 6 of Garble for $R_i = S_j$.
     - The wire labels for S_j's shares of E's input sharings.
     - S_j's shares of the 4 sharings in VerSh.
   - $\mathcal{V}$ matches $\mathcal{O}(\mathsf{GC} \| \mathsf{VerSh})$.
2. **Evaluation.** E follows the evaluation algorithm Eval to evaluate the garbled circuit, except that E uses error correction to reconstruct the secrets (see Section H for the detailed evaluation process).

3. **Sending the Output.** After evaluating the garbled circuits, E reconstructs his own outputs from the output sharings via error correction and sends $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ to G.
4. **Decoding the Output.** G runs the Decode to decode his output $\overline{y}_G$ of $\overline{C}$ Then, G checks whether $\overline{y}_G$ is a valid output of En. If not, G aborts the protocol. Otherwise, G computes his outputs of C by $\mathsf{De}(\overline{y}_G) \oplus \mathsf{mask}$.

Figure 9: The fourth round of our maliciously secure 2PC.

The achieved result is stated below.

Theorem 2 *Assuming parallel oblivious transfers and a random oracle, there is a maliciously secure two-party computation protocol for any Boolean circuit C with communication of $O(|C| \log T + D(\log T + \kappa)^2 \log T + \mathsf{poly}(\kappa, \log T))$ bits plus $O(W_I + \mathsf{poly}(\kappa, \log T))$ invocations of OTs with message length $O(\log T)$ bits against any computationally unbounded adversary with T random oracle queries, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the statistical security parameter. The protocol achieves statistical security with error $2^{-\kappa}$ and only requires 1 OT round and 3 one-way communication rounds.*

We provide the security proof in Section I and give a detailed cost analysis in Section J.

Moreover, in the case where we assume random bit-OT correlations (which is also considered in the concurrent work [IJPS26]) or assume a semi-malicious 2-round OT with adaptive input choices, we are able to achieve a 3-round protocol with the same communication. The basic idea is to use the 2-round OT protocol (which can be constructed from random bit-OT correlations) to instantiate the parallel OTs, and the first-round OT message is sent together with the authentication message of the evaluator's OT inputs (via Fiat-Shamir [FS86]) in the first round of the protocol. In the second round, the garbler sends the second-round OT messages, the garbled circuit, and the authentication messages of the garbled circuit (still via Fiat-Shamir). Finally, the evaluator sends the garbler's masked output to the garbler in the last round. This approach can be further generalized if we assume a 2-round bit-OT protocol (in the plain model) instead of the random bit-OT correlations.

If only the evaluator has output, the last round is no longer required, so we can get a 2-round protocol, i.e. a non-interactive secure computation (NISC) protocol, in this case. We restate the results below and refer the readers to Section K for more details.

Theorem 3 *Assuming $O(W_I \cdot \log T)$ random bit-OT correlations and a random oracle, there is a maliciously secure non-interactive secure computation (NISC) protocol for any Boolean circuit C with communication of $O(|C| \log T + D(\log T + \kappa)^2 \log T + \mathsf{poly}(\kappa, \log T))$ bits against any computationally unbounded adversary with T random oracle queries, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the statistical security parameter. The protocol achieves statistical security with error $2^{-\kappa}$.*

Theorem 4 *For any function $f(\kappa) = \omega(\log \kappa)$, assuming a semi-malicious 2-round OT protocol (with message length $O(f(\kappa))$) with adaptive input choices in the CRS model (that achieves the computational security with advantage $2^{-\kappa}$ with $\mathsf{CC}_{\mathsf{OT}}(\kappa)$ communication cost) and a random oracle, there is a maliciously secure non-interactive secure computation protocol for any Boolean circuit C with communication of $O(|C| f(\kappa) + D\kappa^2 f(\kappa) + \mathsf{poly}(\kappa) + W_I \mathsf{CC}_{\mathsf{OT}}(O(\kappa)))$ bits against any probabllistic polynomial-time adversary, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the computational security parameter. The protocol achieves computational security with advantage $2^{-\kappa}$.*

References

[BCG+18] Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishability obfuscation for ram programs and succinct randomized encodings. *SIAM Journal on Computing*, 47(3):1123–1210, 2018.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556, 2014.

[BGH+23] Gabrielle Beck, Aarushi Goel, Aditya Hegde, Abhishek Jain, Zhengzhong Jin, and Gabriel Kaptchuk. Scalable multiparty garbling. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023*, pages 2158–2172. ACM, 2023.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *the ACM Conference on Computer and Communications Security, CCS'12*, pages 784–796. ACM, 2012.

[BK24] Chunghun Baek and Taechan Kim. Can we beat three halves lower bound?: (Im)possibility of reducing communication cost for garbled circuits. Cryptology ePrint Archive, Report 2024/803, 2024.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 503–513. ACM, 1990.

[CC06] Hao Chen and Ronald Cramer. Algebraic geometric secret sharing schemes and secure multiparty computations over small fields. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference*, volume 4117 of *Lecture Notes in Computer Science*, pages 521–536. Springer, 2006.

[CCXY18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference*, volume 10993 of *Lecture Notes in Computer Science*, pages 395–426. Springer, 2018.

[CHHK25a] Geoffroy Couteau, Carmit Hazay, Aditya Hegde, and Naman Kumar. Breaking the $1/\lambda$-rate barrier for arithmetic garbling. In *Advances in Cryptology – EUROCRYPT 2025, Part VI*, volume 15606 of *Lecture Notes in Computer Science*, pages 182–213, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[CHHK25b] Geoffroy Couteau, Carmit Hazay, Aditya Hegde, and Naman Kumar. $\omega(1/\lambda)$-rate boolean garbling scheme from generic groups. *CRYPTO 2025*, 2025.

[COPZ25] Kelong Cong, Emmanuela Orsini, Erik Pohle, and Oliver Zajonc. Row reduction techniques for n-party garbling. *CRYPTO 2025*, 2025.

[CV22] Michele Ciampi and Ivan Visconti. Efficient NIZK arguments with straight-line simulation and extraction. In *CANS 22: 21st International Conference on Cryptology and Network Security*, volume 13641 of *Lecture Notes in Computer Science*, pages 3–22, Dubai, United Arab Emirates, November 13–15, 2022. Springer, Cham, Switzerland.

[CWYY23] Hongrui Cui, Xiao Wang, Kang Yang, and Yu Yu. Actively secure half-gates with minimum overhead under duplex networks. In *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 35–67, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2005.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2007.

[FJN+13] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 537–556, Athens, Greece, May 26–30, 2013. Springer Berlin Heidelberg, Germany.

[FJNT15] Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. TinyLEGO: An interactive garbling scheme for maliciously secure two-party computation. Cryptology ePrint Archive, Report 2015/309, 2015.

[FLZ24] Lei Fan, Zhenghao Lu, and Hong-Sheng Zhou. Column-wise garbling, and how to go beyond the linear model. Cryptology ePrint Archive, Report 2024/415, 2024.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1986. Springer Berlin Heidelberg, Germany.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th Annual ACM Symposium on Theory of Computing*, pages 699–710, Victoria, BC, Canada, May 4–6, 1992. ACM Press.

[GIS18] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-round MPC: Information-theoretic and black-box. In *TCC 2018: 16th Theory of Cryptography Conference, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 123–151, Panaji, India, November 11–14, 2018. Springer, Cham, Switzerland.

[GKP+13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. STOC '13, page 555–564, 2013.

[GLM+24] Vipul Goyal, Junru Li, Ankit Kumar Misra, Rafail Ostrovsky, Yifan Song, and Chenkai Weng. Dishonest majority constant-round mpc with linear communication from ddh. In *Advances in Cryptology – ASIACRYPT 2024*, pages 167–199, 2024.

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 567–578. ACM Press, 2015.

[GLOS25] Vipul Goyal, Junru Li, Rafail Ostrovsky, and Yifan Song. Towards building scalable constant-round MPC from minimal assumptions via round collapsing. *CRYPTO 2025*, 2025.

[GPS21] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Unconditional communication-efficient MPC via hall's marriage theorem. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021*, volume 12826 of *Lecture Notes in Computer Science*, pages 275–304. Springer, 2021.

[GPS22] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022*, volume 13510 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2022.

[GS96] Arnaldo Garcia and Henning Stichtenoth. On the asymptotic behaviour of some towers of function fields over finite fields. *Journal of Number Theory*, 61(2):248–273, 1996.

[HIV17] Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Actively secure garbled circuits with constant communication overhead in the plain model. In *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 3–39, 2017.

[HKN+25] David Heath, Vladimir Kolesnikov, Varun Narayanan, Rafail Ostrovsky, and Akash Shah. Multiparty garbling from OT with linear scaling and RAM support. *CRYPTO 2025*, 2025.

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 415–434, 2023.

[HSS17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *Advances in Cryptology – ASIACRYPT 2017*, volume 10624 of *Lecture Notes in Computer Science*, pages 598–628. Springer, 2017.

[IJPS26] Yuval Ishai, Ziyang Jin, Naty Peter, and Akshayaram Srinivasan. Non-interactive secure computation with constant communication overhead. In *EUROCRYPT 2026*, 2026.

[ILL25] Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling from homomorphic secret sharing. *CRYPTO 2025*, 2025.

[JRR25] Jake Januzelli, Mike Rosulek, and Lawrence Roy. Lower bounds for garbled circuits from shannon-type information inequalities. *CRYPTO 2025*, 2025.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. STOC '15, page 419–428, 2015.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. Flexor: Flexible garbling for XOR gates that beats free-xor. In *Advances in Cryptology - CRYPTO 2014*, volume 8617 of *Lecture Notes in Computer Science*, pages 440–457. Springer, 2014.

[KNR+17] Vladimir Kolesnikov, Jesper Buus Nielsen, Mike Rosulek, Ni Trieu, and Roberto Trifiletti. DUPLO: Unifying cut-and-choose for garbled circuits. In *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 3–20. ACM Press, 2017.

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 365–391, 2018.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[LGW24] Ruiyang Li, Chun Guo, and Xiao Wang. Towards optimal garbled circuits in the standard model. Cryptology ePrint Archive, Report 2024/1907, 2024.

[LLL25] Hanjun Li, Huijia Lin, and George Lu. Succinct garbled circuits with low-depth garbling algorithms. Cryptology ePrint Archive, Paper 2025/2308, 2025.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78, 2007.

[LS25a] Junru Li and Yifan Song. Constant-round asynchronous MPC with optimal resilience and linear communication. *CRYPTO 2025*, 2025.

[LS25b] Junru Li and Yifan Song. Honest majority constant-round MPC with linear communication from one-way functions. *TCC 2025*, 2025.

[LWYY25a] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. Authenticated bitgc for actively secure rate-one 2pc. In *Advances in Cryptology – CRYPTO 2025*, pages 652–687, Cham, 2025. Springer Nature Switzerland.

[LWYY25b] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. BitGC: Garbled circuits with 1 bit per gate. In *Advances in Cryptology – EUROCRYPT 2025, Part VII*, volume 15607 of *Lecture Notes in Computer Science*, pages 437–466, 2025.

[MORS24] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret sharing. In *TCC 2024: 22nd Theory of Cryptography Conference, Part IV*, volume 15367 of *Lecture Notes in Computer Science*, pages 71–97, 2024.

[MORS25] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. *CRYPTO 2025*, 2025.

[MPP20] Andrew Morgan, Rafael Pass, and Antigoni Polychroniadou. Succinct non-interactive secure computation. In *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 216–245, 2020.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 368–386, 2009.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the First ACM Conference on Electronic Commerce (EC-99)*, pages 129–139. ACM, 1999.

[NST17] Jesper Buus Nielsen, Thomas Schneider, and Roberto Trifiletti. Constant round maliciously secure 2PC with function-independent preprocessing using LEGO. In *ISOC Network and Distributed System Security Symposium – NDSS 2017*, 2017.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *Advances in Cryptology - ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2009.

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In *59th Annual Symposium on Foundations of Computer Science*, pages 859–870. IEEE Computer Society Press, 2018.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? beating the half-gates lower bound for garbled circuits. In *Advances in Cryptology - CRYPTO 2021*, volume 12825 of *Lecture Notes in Computer Science*, pages 94–124. Springer, 2021.

[WB86] Lloyd R Welch and Elwyn R Berlekamp. Error correction for algebraic block codes, December 30 1986. US Patent 4,633,470.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 39–56. ACM, 2017.

[XHX25] Fei Xu, Honggang Hu, and Changhong Xu. Bitwise garbling schemes — a model with $\frac{3}{2}\kappa$-bit lower bound of ciphertexts. *CRYPTO 2025*, 2025.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164. IEEE Computer Society, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.

[YWZ20] Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1627–1646. ACM, 2020.

[ZLC+08] Zhifang Zhang, Mulan Liu, Yeow Meng Chee, San Ling, and Huaxiong Wang. Strongly multiplicative and 3-multiplicative linear secret sharing schemes. In *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2008.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology - EUROCRYPT 2015*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250. Springer, 2015.

A An Overview for [GLOS25]

In this section, we give an overview of how the work [GLOS25] achieves a constant-size ($O(|C|\kappa + D \cdot \mathsf{poly}(n, \kappa))$) multiparty garbled circuit. The main idea is to garble the next-message functions of an underlying protocol, ensuring that the outputs of the garbled circuits in each round are sufficient for computing the input labels for the garbled circuits of the next round.

The Underlying Protocol. We first introduce the structure of the underlying protocol compiled in [GLOS25]. In particular, the protocol is constructed in the setting of $n > 4t$ and is based on a secret sharing scheme Σ (denoted by $[\cdot]$) with a secret size $k = \Theta(n)$ and a constant share size [CC06]. Such a secret sharings scheme is linear over $\mathbb{F}_2$. Like the standard packed Shamir secret sharing scheme [FY92], the local tensor products of the shares of two such sharings still form a similar linear secret sharing $\Sigma^{(2)}$ (denoted by $[\cdot]^{(2)}$, similar for $[\cdot]^{(3)}$ that denotes the product of a $\Sigma^{(2)}$-sharing and a Σ-sharing), where the only difference is that we need more shares for reconstruction. We denote the product of For simplicity, we introduce the underlying protocol for a SIMD circuit that computes k^2 copies of the same circuit together. Such a protocol can be generated to support general circuits using the network routing technique [GPS21, GPS22].

In general, the protocol is constructed in a gate-by-gate manner. For each group of addition/multiplication gates (over $\mathbb{F}_2$), given the two input sharings $[\boldsymbol{x}], [\boldsymbol{y}]$, compute the output sharing $[\boldsymbol{z}]$, where $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}$ for addition gates and $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$ for multiplication gates. Since Σ is $\mathbb{F}_2$-linear, addition gates can be evaluated via local computation: Each party simply adds up his two local shares. For multiplication gates, utilizing the multiplicative property of Σ, $[\boldsymbol{z}]^{(2)} = [\boldsymbol{x}] * [\boldsymbol{y}]$ can be locally computed. Now it is sufficient to transform $[\boldsymbol{z}]^{(2)}$ to $[\boldsymbol{z}]$.

To perform this transformation, the idea of [GLOS25] is to transform a batch of k $\Sigma^{(2)}$-sharings each time. Suppose the input sharings are denoted by $[\boldsymbol{z}_1]^{(2)}, \ldots, [\boldsymbol{z}_k]^{(2)}$. The secrets can be viewed as a matrix $\boldsymbol{Z}$ of size $k \times k$ and the i-th row of $\boldsymbol{Z}$ is $\boldsymbol{Z}_i = \boldsymbol{z}_i$. We use $\boldsymbol{Z}_{\star,j}$ to denote the j-th column of $\boldsymbol{Z}$. The solution is to first let all parties obtain $[\boldsymbol{Z}_{\star,1}], \ldots, [\boldsymbol{Z}_{\star,n}]$ only using reconstruction of $\Sigma^{(2)}$-sharings. Since the sharing algorithm $\Sigma.\mathsf{Sh}$ of Σ is $\mathbb{F}_2$-linear, we may view it as a linear circuit and securely compute this circuit over the input $[\boldsymbol{z}_1]^{(2)}, \ldots, [\boldsymbol{z}_k]^{(2)}$. In this way, all servers can obtain $\Sigma^{(2)}$-sharing of the shares of each party. Then all servers reconstruct the shares to each party. To be more concrete,

1. All servers first prepare random $\Sigma^{(2)}$-sharings $[\boldsymbol{r}_1]^{(2)}, \ldots, [\boldsymbol{r}_{n\ell}]^{(2)}$. We will compute $\Sigma.\mathsf{Sh}(\boldsymbol{Z}_{\star,j}, (r_{1,j}, \ldots, r_{n\ell,j}))$ for all j.
2. All servers view $\Sigma.\mathsf{Sh}$ as a linear circuit. Let $\boldsymbol{y}_{\star,j} = \Sigma.\mathsf{Sh}(\boldsymbol{Z}_{\star,j}, (r_{1,j}, \ldots, r_{n\ell,j}))$ which is of size $n\ell$. Then all servers can locally compute $[\boldsymbol{y}_i]^{(2)}$, where $\boldsymbol{y}_i = (y_{i,1}, \ldots, y_{i,k})$. All parties reconstruct $\boldsymbol{y}_i$ to the $\lceil i/\ell \rceil$-th party. As a result, all parties obtain $[\boldsymbol{Z}_{\star,1}], \ldots, [\boldsymbol{Z}_{\star,k}]$.

To complete the transformation and obtain $[\boldsymbol{z}_1], \ldots, [\boldsymbol{z}_k]$, we only need to apply the above process on $[\boldsymbol{Z}_{\star,1}], \ldots, [\boldsymbol{Z}_{\star,k}]$ again.

As demonstrated, the evaluation phase contains only reconstructions of $\Sigma^{(2)}$-sharings, and each party's local computations depend exclusively on the secrets contained in these sharings and the preprocessed/input data. Furthermore, the $\Sigma^{(2)}$-sharings to be reconstructed are public sharings that can be known by all the parties. This is an important property utilized by [GLOS25]. In particular, the above protocol for a general circuit C only requires communication of $O(|C| + D \cdot n^2)$ bits and $O(|C| + D \cdot n^2\kappa)$ AND operations in the parties' local computations.

Round Collapsing via Garbling Local Computations. The starting point of [GLOS25] is to garble the local computations of all the parties during the evaluation phase of the underlying protocol and send the garbled circuits to an evaluator. However, merely possessing the garbled circuit is insufficient. The evaluator should also be able to input wire labels for the secrets of the $\Sigma^{(2)}$-sharings $[\boldsymbol{z}]^{(2)}$ that need to be reconstructed during the protocol.

The core technique for preparing the wire labels $k_{w,0}$ and $k_{w,1}$ (corresponding to wire w carrying each bit of the secret $\boldsymbol{z}$) involves the following initial setup: the evaluator distributes $[\boldsymbol{k_0}]^{(3)}$ and $[\boldsymbol{k_1} - \boldsymbol{k_0}]$ (where

$\boldsymbol{k_0}$ is the vector containing $k_{w,0}$ for the wires carring all the bits of $\boldsymbol{z}$) during the preprocessing phase. Given $[\boldsymbol{z}]^{(2)}$, parties can then locally compute the label's sharing via:

$$[\boldsymbol{k_z}]^{(3)} = [\boldsymbol{k_0}]^{(3)} + [\boldsymbol{z}]^{(2)} \otimes [\boldsymbol{k_1} - \boldsymbol{k_0}].$$

Crucially, recall that in the underlying protocol, all reconstructed $\Sigma^{(2)}$-sharings are public. Consequently, the evaluator learns these sharings $[\boldsymbol{z}]^{(2)}$ during the garbled circuit evaluation. This enables the evaluator to reconstruct both the underlying secret $\boldsymbol{z}$ and its corresponding wire label $\boldsymbol{k_z}$. With this information, the evaluator can simulate the entire execution of the underlying protocol.

Now, let's analyze the communication cost of the above approach. For each reconstruction in the underlying protocol, the receiver needs to distribute two sharings for the label (whose size is $O(\kappa)$) of the secret. The communication cost of these steps is close to κ times the communication cost of the underlying protocol. Given that the communication cost of protocol is $O(|C| + D \cdot n^2)$ (as we need to reconstruct $[\boldsymbol{z}]^{(2)}$ to all the parties for each multiplication gate), the communication cost of distributing these sharings is $O(|C|\kappa + D \cdot n^2\kappa)$. The other cost mainly comes from sending the garbled circuit. Since we can apply the FreeXOR technique [KS08] in the random oracle model, it only relies on the number of local AND operations in the underlying protocol. Given that the number of local AND operations in the underlying protocol is also $O(|C| + D \cdot n^2)$, the total garbled circuit size is $O(|C|\kappa + D \cdot n^2\kappa)$ using FreeXOR [KS08].

B Formal Definitions of Linear Secret Sharing Schemes

We give the formal definitions of linear secret sharing schemes (LSSSs) here.

Definition 2 (Projection Maps). *Let $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in (\mathbb{F}_q^\ell)^n$. Let $A \subset \{1, \ldots, n\}$ be a non-empty set. The projection map $\pi_A : (\mathbb{F}_q^\ell)^n \to (\mathbb{F}_q^\ell)^{|A|}$ is defined by $\pi_A(\boldsymbol{x}) = (\boldsymbol{x}_i)_{i \in A}$.*

Definition 3 (Linear Secret Sharing Schemes) *Let $\mathbb{F}_q$ be a finite field, and let k, ℓ, and $t < n$ be positive integers. An (n, t, k, ℓ)-linear secret sharing scheme (LSSS) Σ over $\mathbb{F}_q$ consists of two deterministic algorithms $\Sigma.\mathsf{Sh}(\cdot,\cdot) : \mathbb{F}_q^k \times \mathbb{F}_q^{n\ell} \to (\mathbb{F}_q^\ell)^n$ and $\Sigma.\mathsf{Rec}(\cdot) : (\mathbb{F}_q^\ell)^n \to \mathbb{F}_q^k$. For every $\boldsymbol{s} \in \mathbb{F}_q^k$ and $\boldsymbol{r} \in \mathbb{F}_q^{n\ell}$, $\Sigma.\mathsf{Sh}(\boldsymbol{s}, \boldsymbol{r})$ is a linear function that outputs a vector of shares $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) \in (\mathbb{F}_q^\ell)^n$. For any $\boldsymbol{c} \in (\mathbb{F}_q^\ell)^n$ which can be outputted by $\Sigma.\mathsf{Sh}(\boldsymbol{s}, \boldsymbol{r})$ for some $\boldsymbol{r} \in \mathbb{F}_q^{n\ell}$, we call $\boldsymbol{c}$ a Σ-sharing of $\boldsymbol{s}$. We require the following three properties.*

- *t-privacy: For all $\boldsymbol{s}, \boldsymbol{s}' \in \mathbb{F}_q^k$ and $A \subset \{1, \ldots, n\}$ of size $\leq t$,*

$$\{\boldsymbol{r} \xleftarrow{\$} \mathbb{F}_q^{n\ell}, \boldsymbol{c} \leftarrow \Sigma.\mathsf{Sh}(\boldsymbol{s}, \boldsymbol{r}) : \pi_A(\boldsymbol{c})\} \equiv \{\boldsymbol{r}' \xleftarrow{\$} \mathbb{F}_q^{n\ell}, \boldsymbol{c}' \leftarrow \Sigma.\mathsf{Sh}(\boldsymbol{s}', \boldsymbol{r}') : \pi_A(\boldsymbol{c}')\}.$$

- *Reconstruction: For every $\boldsymbol{s} \in \mathbb{F}_q^k$, it holds that for any Σ-sharing $\boldsymbol{c}$ of $\boldsymbol{s}$, $\Sigma.\mathsf{Rec}(\boldsymbol{c}) = \boldsymbol{s}$.*
- *Linearity: The two algorithms $\Sigma.\mathsf{Sh}(\cdot,\cdot) : \mathbb{F}_q^k \times \mathbb{F}_q^{n\ell} \to (\mathbb{F}_q^\ell)^n$ and $\Sigma.\mathsf{Rec}(\cdot) : \mathbb{F}_q^{n\ell} \to \mathbb{F}_q^k$ are both $\mathbb{F}_q$-linear.*

The multiplicative properties of an LSSS is defined as follows.

Definition 4 ([GLOS25]). *We say an (n, t, k, ℓ)-LSSS Σ has d-multiplicative reconstruction if there exists (n, t, k, ℓ^α)-LSSSs $\Sigma^{(\alpha)}$ for $\alpha = 2, \ldots, d$ such that:*

- *For any $\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)} \in \mathbb{F}_q^k$ and Σ-sharings $\boldsymbol{c}^{(1)} = (\boldsymbol{c}_1^{(1)}, \ldots, \boldsymbol{c}_n^{(1)})$, $\boldsymbol{c}^{(2)} = (\boldsymbol{c}_1^{(2)}, \ldots, \boldsymbol{c}_n^{(2)})$ of $\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}$ respectively, $(\boldsymbol{c}_1^{(1)} \otimes \boldsymbol{c}_1^{(2)}, \ldots, \boldsymbol{c}_n^{(1)} \otimes \boldsymbol{c}_n^{(2)})$ is a $\Sigma^{(2)}$-sharing of $\boldsymbol{s}^{(1)} * \boldsymbol{s}^{(2)}$.*
- *For any $\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)} \in \mathbb{F}_q^k$, $\Sigma^{(j-1)}$-sharing $\boldsymbol{c}^{(1)} = (\boldsymbol{c}_1^{(1)}, \ldots, \boldsymbol{c}_n^{(1)})$ of $\boldsymbol{s}^{(1)}$, and Σ-sharing $\boldsymbol{c}^{(2)}$ of $\boldsymbol{s}^{(2)}$, $(\boldsymbol{c}_1^{(1)} \otimes \boldsymbol{c}_1^{(2)}, \ldots, \boldsymbol{c}_n^{(1)} \otimes \boldsymbol{c}_n^{(2)})$ is a $\Sigma^{(j)}$-sharing of $\boldsymbol{s}^{(1)} * \boldsymbol{s}^{(2)}$ for each $j = 3, \ldots, d$.*

Remark 3 *As noted in [GLOS25], for an (n, t, k, ℓ)-LSSS Σ, there are algorithms that can efficiently determine whether a set S of shares comes from a valid Σ-sharing and give such a sharing (if it is valid). There are also algorithms that can efficiently sample a set of at most t_1 shares from another t_2 share ($t_1 + t_2 \leq T$) of a Σ-sharing without the secret. We refer the readers to [GLOS25] for more details.*

C The Formal Security Requirement for the Underlying Protocol

We give the formal security requirement for the underlying protocol Π_0 here. Suppose that the reconstruction process needs to be done rec times during the evaluation phase. Let the $\Sigma^{(2)}$-sharings to be reconstructed be $[\boldsymbol{s}_1]^{(2)}, \ldots, [\boldsymbol{s}_{\mathsf{rec}}]^{(2)}$. We require Π_0 to satisfy the following variant of semi-honest security:

- In the real world, a semi-honest adversary $\mathcal{A}$ corrupts a subset of clients and a subset $\mathcal{C}$ of servers. On input $\boldsymbol{x}$, the output in the real-world execution is defined by

$$\mathsf{REAL}_{\Pi_0,\mathcal{A},\mathcal{C}}(\boldsymbol{x}) = (\mathsf{View}_{\mathcal{C}}(\boldsymbol{x}), ([\boldsymbol{s}_1]^{(2)}, \ldots, [\boldsymbol{s}_{\mathsf{rec}}]^{(2)}), \mathsf{output}^{\Pi_0}(\boldsymbol{x})),$$

  where $\mathsf{View}_{\mathcal{C}}(\boldsymbol{x})$ means corrupted parties' view, and $\mathsf{output}^{\Pi_0}(\boldsymbol{x})$ means the outputs of the honest clients obtained from the execution of Π_0.

- In the ideal world, an ideal adversary Sim_0 controls the same subset of clients and servers. Sim_0 also has one-time access to the ideal functionality $\mathcal{F}$. The output in the ideal-world execution is defined to be the collection of the output of Sim_0 and the outputs of the clients, denoted by $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}_0,\mathcal{C}}(\boldsymbol{x})$. For any ordered list $\mathcal{L} = (S_{i_1}, \ldots, S_{i_m})$ of all elements in $\mathcal{C}$, we further require that Sim_0 can be performed by first computing the views of corrupted clients together with the $\Sigma^{(2)}$-sharings to be reconstructed, then computing the view of S_{i_1}, and then computing each S_{i_j}'s view based on the views of $\{S_{i_1}, \ldots, S_{i_{j-1}}\}$ (without knowing $\mathcal{C}$).

We require that for all subsets of corrupted clients and all sets $\mathcal{C}$ that may contain at most t servers, there exists a PPT adversary Sim_0 meeting the above requirements such that

$$\mathsf{REAL}_{\Pi_0,\mathcal{A},\mathcal{C}}(\boldsymbol{x}) \equiv \mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}_0,\mathcal{C}}(\boldsymbol{x}).$$

Remark 4 *We need two additional requirements on the underlying protocol over the standard semi-honest security:*

- *The first requirement is that the $\Sigma^{(2)}$-sharings reconstructed in the evaluation phase can be made public. This can be easily realized by additionally generating a random $\Sigma^{(2)}$-sharing with respect to each $\Sigma^{(2)}$-sharing to be reconstructed in $\mathcal{F}_{\mathsf{prep}}$, where the receiver of the $\Sigma^{(2)}$-sharing holds the secret. Then, the $\Sigma^{(2)}$-sharing to be reconstructed can be masked by the random sharing, and we only need the receiver to take an additional layer of XOR gates in his local computation.*

- *The other requirement is that the views of corrupted servers can be generated one by one. This is a weaker version of semi-honest adaptive security, where the adaptive adversary should choose the corrupted clients at the beginning, and he can only adaptively corrupt a set of $\leq t$ servers at the end of the protocol.*

D An Instantiation of the Underlying Protocol

The underlying protocols Π_0, Π_0' are slightly modified from the underlying protocol in [GLOS25]. We refer the readers to [GLOS25] for an overview of it. Here, we directly provide the protocol. We first provide the protocol description of Π_0 and show that it satisfies the requirements for our garbling scheme. Then, we will present how the protocol is modified to Π_0' for our 2PC. For a matrix $\mathcal{M}$, let $\mathcal{M}_{[:,i]}$ denote its i-th column. For a list $\mathcal{L}$, let $\mathcal{L}[i]$ denote its i-th entry.

D.1 Subprotocols for Π_0

We underlying protocol Π_0 is built upon a (n, t, k, ℓ)-LSSS with 3-multiplicative over $\mathbb{F}_2$. The protocol is constructed following the technique of network routing [GPS21, GPS22], which reduces the task of building an MPC upon an LSSS to the tasks of evaluating batches of multiplication gates, performing sharing transformations, and secret collections. We will provide the subprotocols for these tasks, and we will also provide a subprotocol of circuit preprocessing that is required for the network routing technique.

The Transpose Protocol. We present the formal description of $\Pi_{\mathsf{Transpose}}$. $\Pi_{\mathsf{Transpose}}$ takes sharings $[\boldsymbol{x}_1]^{(2)}, \ldots, [\boldsymbol{x}_k]^{(2)}$ as input. Let $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,k})$ and $\boldsymbol{x}_i^* = (x_{1,i}, \ldots, x_{k,i})$, $\Pi_{\mathsf{Transpose}}$ outputs $([\boldsymbol{x}_1^*], \ldots, [\boldsymbol{x}_k^*])$.

Recall that for an LSSS, the sharing algorithm Sh is linear on each field element of its input, so $\Sigma.\mathsf{Sh}_1, \ldots, \Sigma.\mathsf{Sh}_n : \mathbb{F}_2^{k+n\ell} \to \mathbb{F}_2^{\ell}$ are all $\mathbb{F}_2$-linear functions. Suppose that for each $i = 1, \ldots, n$,

$$\begin{aligned}&\Sigma.\mathsf{Sh}_i((s_1, \ldots, s_k), (a_1, \ldots, a_{n\ell}))\\ =&\left(\sum_{j=1}^{k} c_{1,j}^{(i)} s_j + \sum_{j=1}^{n\ell} c_{1,k+j}^{(i)} a_j, \ldots, \sum_{j=1}^{k} c_{\ell,j}^{(i)} s_j + \sum_{j=1}^{n\ell} c_{\ell,k+j}^{(i)} a_j\right).\end{aligned}$$

Correspondingly, we define $\mathbb{F}_2$-linear functions $F_1, \ldots, F_n : (\mathbb{F}_2^k)^{k+n\ell} \to (\mathbb{F}_2^k)^{\ell}$ by

$$F_i(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k+n\ell}) = \left(\sum_{j=1}^{k+n\ell} c_{1,j}^{(i)} \boldsymbol{v}_j, \ldots, \sum_{j=1}^{k+n\ell} c_{\ell,j}^{(i)} \boldsymbol{v}_j\right)$$

for each $i = 1, \ldots, n$ and give $\Pi_{\mathsf{Transpose}}$ as follows.

Protocol $\Pi_{\mathsf{Transpose}}$

Input: Each server's shares of input sharings $[\boldsymbol{x}_1]^{(2)}, \ldots, [\boldsymbol{x}_k]^{(2)}$. Let each $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,k})$ and set $\boldsymbol{x}_i^* = (x_{1,i}, \ldots, x_{k,i})$.

1. The servers take a group of $\Sigma^{(2)}$ sharings $[\boldsymbol{r}_1]^{(2)}, \ldots, [\boldsymbol{r}_n]^{(2)}$ (where each server S_j holds the secret $\boldsymbol{r}_j$) and $n\ell$ random $\Sigma^{(2)}$-sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ from the output of $\mathcal{F}_{\mathsf{prep}}$ and associate them with the execution of $\Pi_{\mathsf{Transpose}}$.
2. The servers locally compute $[\boldsymbol{y}_i]^{(2)} = [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell}) + \boldsymbol{r}_i]^{(2)}$ for $i = 1, \ldots, n$.
3. For each $i = 1, \ldots, n$, each server sends his share of $[\boldsymbol{y}_i]^{(2)}$ to server S_i for reconstruction.
4. Each server S_j reconstructs $\boldsymbol{y}_j$ from $[\boldsymbol{y}_i]^{(2)}$ and computes the vector of his shares of $([\boldsymbol{x}_1^*], \ldots, [\boldsymbol{x}_k^*])$ by $\boldsymbol{y}_i - \boldsymbol{r}_i$.

Figure 10: The Transpose Protocol.

Evaluating Multiplication Gates. Then, we show how to utilize $\Pi_{\mathsf{Transpose}}$ to compute a group of multiplication gates. Each k groups of k multiplication gates (k^2 multiplication gates in total) are computed together. This can be done with $O(n^2)$-bit communication by the following protocol Π_{Multi}.

Protocol Π_{Multi}

Input: The servers input their shares of Σ-sharings $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_k]$ and $[\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_k]$. Let $\boldsymbol{z}_1 = \boldsymbol{x}_1 * \boldsymbol{y}_1, \ldots, \boldsymbol{z}_k = \boldsymbol{x}_k * \boldsymbol{y}_k$.

1. The servers locally computes their shares of $[\boldsymbol{z}_j]^{(2)} = [\boldsymbol{x}_j] \otimes [\boldsymbol{y}_j]$ for each $j = 1, \ldots, k$.
2. The servers run $\Pi_{\mathsf{Transpose}}$ with input sharings $[\boldsymbol{z}_1]^{(2)}, \ldots, [\boldsymbol{z}_k]^{(2)}$ and get output sharings $[\boldsymbol{z}_1^*], \ldots, [\boldsymbol{z}_k^*]$.
3. The servers locally computes $[\boldsymbol{z}_j^*]^{(2)} = [\mathbf{1}] \otimes [\boldsymbol{z}_j^*]$ for each $j = 1, \ldots, k$, where $[\mathbf{1}]$ is a public Σ-sharing of an all-1 vector.
4. The servers run $\Pi_{\mathsf{Transpose}}$ with input sharings $[\boldsymbol{z}_1^*]^{(2)}, \ldots, [\boldsymbol{z}_k^*]^{(2)}$ and get output sharings $[\boldsymbol{z}_1], \ldots, [\boldsymbol{z}_k]$.

Figure 11: Protocol to compute batched multiplication gates.

Sharing Transformations. A sharing transformation transforms a Σ-sharing $[\boldsymbol{x}]$ to a Σ-sharing of a linear map L of $\boldsymbol{s}$, i.e. $[L(\boldsymbol{x})]$. We also do k sharing transformations together with respect to k linear maps $L^{(1)}, \ldots, L^{(k)}$. This can be done with $O(n^2)$-bit communication by the following protocol Π_{Tran}.

Protocol Π_{Tran}

Input: The servers input public linear maps $L^{(1)}, \ldots, L^{(k)} : \mathbb{F}^k \to \mathbb{F}^k$ and their shares of Σ-sharings $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_k]$,

1. The servers locally computes $[\boldsymbol{x}_j]^{(2)} = [\mathbf{1}] \otimes [\boldsymbol{x}_j]$ for each $j = 1, \ldots, k$, where $[\mathbf{1}]$ is a public Σ-sharing of an all-1 vector.
2. The servers run $\Pi_{\mathsf{Transpose}}$ with input sharings $[\boldsymbol{x}_1]^{(2)}, \ldots, [\boldsymbol{x}_k]^{(2)}$ and get output sharings $[\boldsymbol{x}_1^*], \ldots, [\boldsymbol{x}_k^*]$.
3. Let $\mathcal{M}$ be a $k \times k$ matrix defined by

$$\mathcal{M} = \begin{pmatrix} L^{(1)}(\boldsymbol{x}_1) \\ L^{(2)}(\boldsymbol{x}_2) \\ \vdots \\ L^{(k)}(\boldsymbol{x}_k) \end{pmatrix}.$$

   Since $L^{(1)}, \ldots, L^{(k)}$ are linear, there exist constant vectors $\boldsymbol{c}_1^{(j)}, \ldots, \boldsymbol{c}_k^{(j)} \in \mathbb{F}^k$ such that

$$\mathcal{M}_{[:,j]} = \sum_{i=1}^{k} \boldsymbol{c}_i^{(j)} * \boldsymbol{x}_i^*$$

   for $j = 1, \ldots, k$. The servers agree on public Σ-sharings $\{[\boldsymbol{c}_i^{(j)}]\}_{1 \le i,j \le k}$. Then the servers locally compute

$$[\mathcal{M}_{[:,j]}]^{(2)} = \sum_{i=1}^{k} [\boldsymbol{c}_i^{(j)}] \otimes [\boldsymbol{x}_i^*]$$

   for $j = 1, \ldots, k$.
4. The servers run $\Pi_{\mathsf{Transpose}}$ with input sharings $[\mathcal{M}_{[:,1]}]^{(2)}, \ldots, [\mathcal{M}_{[:,k]}]^{(2)}$ and get output sharings $[L^{(1)}(\boldsymbol{x}_1)], \ldots, [L^{(k)}(\boldsymbol{x}_k)]$.

Figure 12: Protocol for sharing transformations.

Secret Collections from Different Positions. Secret collection is used to generate a Σ-sharing whose secret is collected from different Σ-sharings. With the help of the sharing transformation technique, we only need to handle a simple condition of secret collection where the secret is collected from k different positions of k sharings. If the input sharings are $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_k]$, we will generate Σ-sharing whose secret consists of the first entry of $\boldsymbol{x}_1$, the second entry of $\boldsymbol{x}_2$, ..., and the k-th entry of $\boldsymbol{x}_k$. Similar to multiplication gates and sharing transformations, we do k such collections together. This can be done with $O(n^2)$-bit communication by the following protocol Π_{Collect}.

Protocol Π_{Collect}

Input: The servers input their shares of $[\boldsymbol{x}_{1,i}], \ldots, [\boldsymbol{x}_{k,i}]$ for $i = 1, \ldots, k$ where each $\boldsymbol{x}_{j,i} = (x_{j,i}^{(1)}, \ldots, x_{j,i}^{(k)})$. Let $\boldsymbol{y}_i = (x_{1,i}^{(1)}, \ldots, x_{k,i}^{(k)})$ for each $i = 1, \ldots, k$.

1. For $i = 1, \ldots, k$, the servers locally compute

$$[\boldsymbol{y}_i]^{(2)} = [(x_{1,i}^{(1)}, \ldots, x_{k,i}^{(k)})]^{(2)} = \sum_{j=1}^{k} [\boldsymbol{e}_j] \otimes [\boldsymbol{x}_{j,i}],$$

   where each $[\boldsymbol{e}_j]$ is a public Σ-sharing of a unit vector with the j-th bit equal to 1.
2. The servers run $\Pi_{\mathsf{Transpose}}$ with input sharings $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_k]^{(2)}$ and get output sharings $[\boldsymbol{y}_1^*], \ldots, [\boldsymbol{y}_k^*]$.
3. The servers locally compute $[\boldsymbol{y}_j^*]^{(2)} = [\mathbf{1}] \otimes [\boldsymbol{y}_j^*]$ for each $j = 1, \ldots, k$.

4. The servers run $\Pi_{\mathsf{Transpose}}$ with input sharings $[\boldsymbol{y}_1^*]^{(2)}, \ldots, [\boldsymbol{y}_k^*]^{(2)}$ and get output sharings $[\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_k]$.

Figure 13: Protocol to do secret collections from different positions.

Circuit Preprocessing. The parties will first transform the original circuit C to a circuit C' with the following properties:

- For all input x, $C(x) = C'(x)$.
- C' consists of an input layer, an output layer, D intermediate layers of addition/multiplication gates, where D is the depth of C. For each input wire of the circuit and output wire of an addition/multiplication gate, there is a fan-out gate taking this wire as input and copying it the number of times this wire will be used in future layers so that *every output wire of the input layer and the intermediate layers* is only used once as input wire in the later layers.
- In the input layer and the output layer, the number of input wires attached to each client and the number of output wires attached to each client are multiples of k. In each intermediate layer, the number of addition gates and the number of multiplication gates are multiples of k^2. The number of input wires of the output layer is a multiple of k^2. The number of output wires of the fan-out gates in the input layer and each intermediate layer is also a multiple of k^2.
- In each layer, k gates of the same type are grouped together (for input/output layers, k input/output wires attached to the same client are grouped together). Each group of output wires from the input layer and each intermediate layer serves as the input wires to a group of fan-out gates in this layer. The number of output wires of each group of fan-out gates is a multiple of k.
- Circuit size: $|C'| = O(|C| + Dk^2 + mk)$, where m is the number of clients that provide inputs and D is the depth of C.

The generation is deterministic, and we refer the readers to [GPS21, GLOS25] for more details. Then, we provide the circuit preprocessing protocol Π_{Circprep} as follows.

Protocol Π_{Circprep}

1. Recall that in C', gates that have the same type in each layer are divided into groups of size k. During the later computation, a batch of k wire values is stored in a single Σ-sharing. All parties determine how the wire values should be packed as follows:
   - For the input layer, for each group of k input gates belonging to the same client, the values of the output wires will be stored in a single Σ-sharing.
   - For each group of fan-out gates in the input layer, suppose it takes the wires $(w_1, \ldots, w_k)$ and vector $(n_1, \ldots, n_k)$ as input. Recall that in C', $(w_1, \ldots, w_k)$ are the output wires of a group of gates, n_i is the number of times we need to make copies of w_i, and $n_1 + \cdots + n_k$ is a multiple of k. All parties determine how the output wires should be packed:
     (a) Each party initiates an empty list $\mathcal{L}$. From $i = 1$ to k, each party inserts n_i times of w_i into L.
     (b) Let $h = (n_1 + \cdots + n_k)/k$. From $i = 1$ to h, the i-th output Σ-sharing will contain the values of wires $\mathcal{L}[(i-1) \cdot k + 1], \ldots, \mathcal{L}[i \cdot k]$.
   - For all computation layers, for each group of k multiplication gates or addition gates:
     * The values of the first input wires of these gates will be stored in a single Σ-sharing.
     * The values of the second input wires of these gates will be stored in a single Σ-sharing.
     * The values of the output wires of these gates will be stored in a single Σ-sharing.
   - For each group of fan-out gates in the intermediate layers, the wire values are packed in the same way as those for each fan-out gate in the input layer.
   - For the output layer, for each group of k output gates belonging to the same client, the values of the input wires will be stored in a single Σ-sharing.

2. Let N denote the number of output sharings of the input layer and all intermediate layers. Then the number of input sharings of the output layer and all intermediate layers is also N. The output sharings are labeled by $1, \ldots, N$, and the input sharings are also labeled by $1, \ldots, N$. The servers follow [GPS21] to attach a permutation p_i on the i-th output sharings.

Figure 14: Protocol for the circuit preprocessing.

D.2 Preprocessing and Input Functionalities

In this section, we present the preprocessing functionality $\mathcal{F}_{\mathsf{prep}}$ and the input functionality $\mathcal{F}_{\mathsf{input}}$.

Preprocessing. The preprocessing functionality $\mathcal{F}_{\mathsf{prep}}$ prepares the random sharings required for the executions of $\Pi_{\mathsf{Transpose}}$.

Functionality $\mathcal{F}_{\mathsf{prep}}$

On receiving (prep, C) from all the honest parties:

1. The trusted party transforms C to C'.
2. For each batch of output wires to each client, the trusted party samples a random $\Sigma^{(2)}$-sharing $[\boldsymbol{o}]^{(2)}$ with an all-zero secret.
3. Let W be the number of wires in C', do the following $10W/k^2$ times:
   (a) For each server S_j, the trusted party generates a random $\Sigma^{(2)}$-sharing $[\boldsymbol{r}'_j]^{(2)}$ and a random $\Sigma^{(2)}$-sharing $[\boldsymbol{o}_j]^{(2)}$ with all zero secret. The trusted party distributes them to all the servers. Then, the trusted party sends $[\boldsymbol{r}'_j]^{(2)}$ to S_j.
   (b) The trusted party generates $n\ell$ random $\Sigma^{(2)}$-sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ and distributes it to all the servers.

Figure 15: The preprocessing functionality.

Input for Each Party. The input functionality $\mathcal{F}_{\mathsf{input}}^{P_i}$ helps the client P_i to generate random Σ-sharings for their input.

Functionality $\mathcal{F}_{\mathsf{input}}^{P_i}$

On receiving (input, x_i) from each client P_i, for each batch of k input bits with input values $s_1, \ldots, s_k \in \mathbb{F}_2$ (obtained from x_i, append zeros after x_i to make the input length a multiple of k), the trusted party randomly generates a Σ-sharing $[\boldsymbol{s}]$ based on the secret and distributes $[\boldsymbol{s}]$ to all the servers, where $\boldsymbol{s} = (s_1, \ldots, s_k)$.

Figure 16: The input functionality.

D.3 Protocol Description

Then, we provide our underlying protocol Π_0 for our garbling scheme as follows.

Protocol Π_0

1. Let C be the circuit to compute. The clients and servers first locally compute C' and send (prep, C) to $\mathcal{F}_{\mathsf{prep}}$. Then, the servers receive:
   – A random $\Sigma^{(2)}$-sharing $[\boldsymbol{r}]^{(2)}$ associated with each batch of output gates in C', and the receiver of the output of each group of gates gets the secrets of this sharing.
   – $10W/k^2$ groups of:
   * Random $\Sigma^{(2)}$-sharings $[\boldsymbol{r}'_1]^{(2)}, \ldots, [\boldsymbol{r}'_n]^{(2)}$ and random $\Sigma^{(2)}$-sharings $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_n]^{(2)}$ with all zero

secrets, where each server S_j holds $\boldsymbol{r}_j$.

* Random $\Sigma^{(2)}$-sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$.

The parties locally compute $[\boldsymbol{r}_i]^{(2)} = [\boldsymbol{r}'_i]^{(2)} + [\boldsymbol{o}_i]^{(2)}$ for $i = 1, \ldots, n$. Each group of sharings is associated with an execution of $\Pi_{\mathsf{Transpose}}$, where $\Pi_{\mathsf{Transpose}}$ will be executed for no more than $10W/k^2$ times in Π_0.

2. Each client P_i sends (input, x_i) to $\mathcal{F}_{\mathsf{input}}^{P_i}$, where x_i is the input of P_i to the circuit C.
3. The parties run the evaluation phase as follows.

Evaluation Phase

(a) The servers agree on a public Σ-sharing $[\mathbf{1}]$ where $\mathbf{1} = (1, 1, \ldots, 1)$. Then each Σ-sharing $[\boldsymbol{s}]$ can be locally converted to a $\Sigma^{(2)}$-sharing $[\boldsymbol{s}]^{(2)} = [\mathbf{1}] \otimes [\boldsymbol{s}]$. For $\boldsymbol{e}_1 = (1, 0, 0, \ldots, 0)$, $\boldsymbol{e}_2 = (0, 1, 0, \ldots, 0)$, $\ldots$, $\boldsymbol{e}_k = (0, 0, \ldots, 0, 1)$, all the servers agree on public Σ-sharings $[\boldsymbol{e}_1], \ldots, [\boldsymbol{e}_k]$.

(b) The clients and servers run Π_{Circprep}.

(c) All parties evaluate the circuit layer by layer as follows:

i. **Handling Fan-out Gates:** For each output sharings $[\boldsymbol{x}]$ of a group of gates (or input wires) in the previous layer, let n_i denote the number of times that the i-th secret of $\boldsymbol{x}$ is used in later layers. Then, the fan-out gates for $\boldsymbol{x}$ computes $(n_1 + \cdots + n_k)/k$ linear transformations on $\boldsymbol{x}$. For each k group of k linear transformations $L_1, \ldots, L_k$ on $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$, the servers can run Π_{Tran} to compute the output $[L_1(\boldsymbol{x}_1)], \ldots, [L_k(\boldsymbol{x}_k)]$. Note that the number of output wires of all the fan-out gates in each layer is a multiple of k^2, which matches that each Π_{Tran} can compute sharings on k^2 output wires of fan-out gates.

ii. **Permuting the Secrets:** For each k output sharings $[\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_k]$ of the previous layer, let $p_1, \ldots, p_k$ denote the permutations (expressed as linear maps from $\mathbb{F}_2^k$ to $\mathbb{F}_2^k$) associated with them. All the servers run Π_{Tran} with input $[\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_k]$ and $p_1, \ldots, p_k$. Then the secrets of the input sharings in the next layer come from different positions in the output sharings of previous layers.

iii. **Collecting Secrets from Previous Layers:** For each k input sharings $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_k]$, let $[\boldsymbol{x}_j^{(i)}]$ denote the output sharing from previous layers whose i-th secret is the i-th secret of $q_j(\boldsymbol{x}_j)$ where q_j is a permutation. Since each of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ comes from different positions in the output sharings of the previous layers, there exists output sharings $[\boldsymbol{x}_j^{(1)}], \ldots, [\boldsymbol{x}_j^{(k)}]$ from the output sharings of previous layers for each $j = 1, \ldots, k$. Then, all servers run Π_{Collect} with input sharings $[\boldsymbol{x}_j^{(1)}], \ldots, [\boldsymbol{x}_j^{(k)}]$ for $j = 1, \ldots, k$.

iv. **Permuting the Secrets:** For each k sharings $[q_1(\boldsymbol{x}_1)], \ldots, [q_k(\boldsymbol{x}_k)]$ collected by Π_{Collect}, all the servers run Π_{Tran} with input $[q_1(\boldsymbol{x}_1)], \ldots, [q_k(\boldsymbol{x}_k)]$ and $q_1^{-1}, \ldots, q_k^{-1}$. Then the servers obtain the input sharings $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_k]$ of this layer.

v. **Evaluating Multiplication Gates and Addition Gates:** For each k group of multiplication gates with input sharings $([\boldsymbol{x}_1], [\boldsymbol{y}_1]), \ldots, ([\boldsymbol{x}_k], [\boldsymbol{y}_k])$, all parties run Π_{Multi} with input sharings $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_k]$ and $[\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_k]$. For each group of addition gates with input sharings $[\boldsymbol{x}], [\boldsymbol{y}]$, all parties locally compute $[\boldsymbol{x} + \boldsymbol{y}] = [\boldsymbol{x}] + [\boldsymbol{y}]$. This step doesn't need to be done for the output layer.

4. Finally, the Σ-sharings for each batch of k output wires are obtained by the servers. The servers then use $[\mathbf{1}]$ to transform them to $\Sigma^{(2)}$-sharings for the outputs and then mask them with the $\Sigma^{(2)}$-sharing $[\boldsymbol{o}]^{(2)}$ with all-zero secret associated with this batch of outputs. The servers then output their shares of the resulting $\Sigma^{(2)}$-sharing to the clients who hold the batch of outputs. The clients then reconstruct their outputs from these sharings.

Figure 17: The instantiation of protocol Π_0.

Clearly, the protocol meets our requirements on the structure of the underlying protocol Π_0 in Section 4, so it only suffices to give the security proof of Π_0. The protocol is similar to the underlying protocol in [GLOS25]. $O(n^2)$ gates in the same layer are evaluated together, and each client's $O(n)$ input wires are packed together, resulting in an $O(|C| + Dn^2 + mn)$ communication complexity using the technique of network routing [GPS21, GPS22]. We refer the readers to [GLOS25] for a detailed cost analysis.

D.4 Security Proof

In this section, we give an ideal adversary Sim_0 to show that the protocol Π_0 meets our requirements. We will show that the output in the ideal world is computationally indistinguishable from that in the real world using hybrid arguments.

We give the ideal adversary Sim_0 below.

Simulator Sim_0

Sim_0 samples all the $\Sigma^{(2)}$-sharings reconstructed in the evaluation phase at the beginning and outputs them.

When a client P_i is corrupted, then Sim_0 obtains the inputs and outputs of P_i. Then, Sim_0 follows the protocol to compute the outputs of $\mathcal{F}_{\mathsf{input}}^{P_i}$ and samples the output $\Sigma^{(2)}$-sharings from all the servers based on each batch of outputs of P_i. Then, Sim_0 outputs the view of P_i.

When a server S_j is corrupted, let the $\mathcal{C}$ be the set of servers corrupted before S_j with $|\mathcal{C}| \leq t-1 = n/4-1$, Sim_0 runs the following to generate S_j's view:

1. If $\mathcal{C} = \emptyset$, Sim_0 follows to protocol to compute C' and emulates $\mathcal{F}_{\mathsf{prep}}$ to receive (prep, C) from S_j. For each output sharing from $\mathcal{F}_{\mathsf{prep}}$ except the $\Sigma^{(2)}$-sharings with all-zero secrets, Sim_0 randomly samples S_j's share of it based on the shares for servers in $\mathcal{C}$. In particular, for each output $\Sigma^{(2)}$-sharing from $\mathcal{F}_{\mathsf{prep}}$ whose secret is output to S_j, Sim_0 generates the secret randomly.
2. For each honest client P_i, for each input sharing output from $\mathcal{F}_{\mathsf{input}}^{P_i}$, Sim_0 randomly samples S_j's share based on the shares for servers in $\mathcal{C}$.
3. Sim_0 generates S_j's view during the evaluation phase of Π_0 as follows:

Evaluation Phase

(a) Sim_0 follows the protocol to get public sharings $[\mathbf{1}]$ and $[\boldsymbol{e}_1], \ldots, [\boldsymbol{e}_k]$ and run Π_{prepcirc}.

(b) Sim_0 simulates the evaluation process layer by layer as follows:

i. **Handling Fan-out Gates:** For each execution of Π_{Tran}:

A. Sim_0 follows the protocol to compute corrupted servers' shares of $[\boldsymbol{x}_j]^{(2)}$ for each $j = 1, \ldots, k$ using the received output from $\mathcal{A}$ while emulating $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}$.

B. For the first execution of $\Pi_{\mathsf{Transpose}}$:

1) For each $i = 1, \ldots, n$, Sim_0 follows the protocol to compute S_j's shares of $[F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$).

2) Note that $[\boldsymbol{y}_i]^{(2)}$ has been sampled. Then, Sim_0 computes S_j's share of $[\boldsymbol{r}_i]^{(2)} = [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell}) - \boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$). Sim_0 then computes S_j's share of the corresponding sharing $[\boldsymbol{o}_i]^{(2)} = [\boldsymbol{r}'_i]^{(2)} - [\boldsymbol{r}_i]^{(2)}$.

3) Sim_0 sends the each honest server's share of $[\boldsymbol{y}_j]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) to S_j on behalf of the honest server and reconstructs the secret $\boldsymbol{y}_j$.

4) Sim_0 follows the protocol to compute S_j's shares of $([\boldsymbol{x}^*_1], \ldots, [\boldsymbol{x}^*_k])$ based on the $\boldsymbol{y}_j$ and $\boldsymbol{r}_j$ (in $\Pi_{\mathsf{Transpose}}$), where $\boldsymbol{r}_j = \boldsymbol{r}'_j$ has been generated while emulating $\mathcal{F}_{\mathsf{prep}}$.

C. Sim_0 follows the protocol to compute S_j's shares of $[\mathcal{M}_{[:,j]}]^{(2)}$ for $j = 1, \ldots, k$ based on S_j's outputs from the preprocessing/input functionalities and the secrets of $\Sigma^{(2)}$-sharings reconstructed for him.

D. Sim_0 simulates the second execution of $\Pi_{\mathsf{Transpose}}$ as in Step B. to generate the $\Sigma^{(2)}$-sharing to S_j, the S_j's shares of the $\Sigma^{(2)}$-sharings with all-zero secrets during this execution of $\Pi_{\mathsf{Transpose}}$, and S_j's shares of $([L_1(\boldsymbol{x}_1)], \ldots, [L_k(\boldsymbol{x}_k)])$.

ii. **Permuting the Secrets:** Sim_0 simulates each execution of Π_{Tran} in this step as in **Handling Fan-out Gates**.

iii. **Collecting Secrets from Previous Layers:** For each execution of Π_{Collect}:

A. Sim_0 follows the protocol to S_j's shares of $[\boldsymbol{y}_i]^{(2)}$ for each $j = 1, \ldots, k$.

B. For the first execution of $\Pi_{\mathsf{Transpose}}$:

1) For each $i = 1, \ldots, n$, Sim_0 follows the protocol to compute S_j's shares of $[F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$).

2) Note that Sim_0 computes S_j's share of $[\boldsymbol{r}_i]^{(2)} = [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell}) - \boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$). Sim_0 then computes S_j's share of the corresponding sharing $[\boldsymbol{o}_i]^{(2)} = [\boldsymbol{r}'_i]^{(2)} - [\boldsymbol{r}_i]^{(2)}$.
3) Sim_0 sends the each honest server's share of $[\boldsymbol{y}_j]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) to S_j on behalf of the honest server and reconstructs the secret $\boldsymbol{y}_j$.
4) Sim_0 follows the protocol to compute S_j's shares of $([\boldsymbol{y}^*_1], \ldots, [\boldsymbol{y}^*_k])$ (in Π_{Collect}) based on the $\boldsymbol{y}_j$ and $\boldsymbol{r}_j$ (in $\Pi_{\mathsf{Transpose}}$), where $\boldsymbol{r}_j = \boldsymbol{r}'_j$ has been generated while emulating $\mathcal{F}_{\mathsf{prep}}$.

C. Sim_0 follows the protocol to compute S_j's shares of $[\boldsymbol{y}^*_j]^{(2)}$ for $j = 1, \ldots, k$.

D. Sim_0 simulates the second execution of $\Pi_{\mathsf{Transpose}}$ as in Step B. to generate the $\Sigma^{(2)}$-sharing to S_j, the S_j's shares of the $\Sigma^{(2)}$-sharings with all-zero secrets during this execution of $\Pi_{\mathsf{Transpose}}$, and S_j's shares of $([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_k])$.

iv. **Permuting the Secrets:** Sim_0 simulates each execution of Π_{Tran} in this step as in **Handling Fan-out Gates**.

v. **Evaluating Multiplication Gates and Addition Gates:** For each group of addition gates, Sim_0 follows the protocol to compute the corrupted servers' shares of the Σ-sharings for output wires. For each execution of Π_{Multi}:

A. Sim_0 follows the protocol to compute S_j's shares of $[\boldsymbol{z}_j]^{(2)}$ for each $j = 1, \ldots, k$.

B. For the first execution of $\Pi_{\mathsf{Transpose}}$:
1) For each $i = 1, \ldots, n$, Sim_0 follows the protocol to compute S_j's shares of $[F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$).
2) Note that $[\boldsymbol{y}_i]^{(2)}$ has been sampled. Then, Sim_0 computes S_j's share of $[\boldsymbol{r}_i]^{(2)} = [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell}) - \boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$). Sim_0 then computes S_j's share of the corresponding sharing $[\boldsymbol{o}_i]^{(2)} = [\boldsymbol{r}'_i]^{(2)} - [\boldsymbol{r}_i]^{(2)}$.
3) Sim_0 sends the each honest server's share of $[\boldsymbol{y}_j]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) to S_j on behalf of the honest server and reconstructs the secret $\boldsymbol{y}_j$.
4) Sim_0 follows the protocol to compute S_j's shares of $([\boldsymbol{z}^*_1], \ldots, [\boldsymbol{z}^*_k])$.

C. Sim_0 follows the protocol to compute S_j's shares of $[\boldsymbol{z}^*_j]^{(2)}$ for $j = 1, \ldots, k$.

D. Sim_0 simulates the second execution of $\Pi_{\mathsf{Transpose}}$ as in Step B. to generate the $\Sigma^{(2)}$-sharing to S_j, the S_j's shares of the $\Sigma^{(2)}$-sharings with all-zero secrets during this execution of $\Pi_{\mathsf{Transpose}}$, and S_j's shares of $([\boldsymbol{z}_1], \ldots, [\boldsymbol{z}_k])$.

4. For each batch of output wires to a corrupted client P_i, Sim_0 follows the protocol to compute S_j's share of the Σ-sharing for this batch of outputs and uses $[\mathbf{1}]$ to transform it to S_j's share of the $\Sigma^{(2)}$-sharing for the output values. Finally, Sim_0 computes S_j's share of the $\Sigma^{(2)}$-sharing $[\boldsymbol{o}]^{(2)}$ with an all-zero secret associated with this batch of outputs based on the $\Sigma^{(2)}$-sharing for the output values and the output $\Sigma^{(2)}$-sharing sampled while generating P_i's view.

5. Sim_0 finally outputs S_j's view.

Figure 18: The simulator for Π_0.

We construct the following hybrids:

Hyb$_0$: In this hybrid, Sim_0 gets honest clients' inputs and runs the protocol honestly. This corresponds to the real-world scenario.

Hyb$_1$: In this hybrid, while **Handling Fan-out Gates** during the evaluation phase, for the first execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Tran}, after each S_j is newly corrupted, Sim_0 doesn't follow the protocol to compute S_j and the remaining honest servers' shares of $[\boldsymbol{y}_i]^{(2)}$ for each server S_i who is honest before S_j is corrupted ($S_i \notin \mathcal{C}$). Instead, Sim_0 first follows the protocol to compute the shares for servers in $\mathcal{C}$ and then randomly samples the whole sharing based on these servers' shares. In addition, Sim_0 does not follow the protocol to emulate $\mathcal{F}_{\mathsf{prep}}$ to generate S_j and the remaining honest servers' shares of the corresponding sharing $[\boldsymbol{o}_i]^{(2)}$. Instead, Sim_0 computes these servers' shares of $[\boldsymbol{o}_i]^{(2)}$ by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)} - [\boldsymbol{r}'_i]^{(2)}$ for each server $S_i \notin \mathcal{C}$.

Similarly, Sim_0 doesn't follow the protocol to compute honest servers' shares of $[\boldsymbol{y}_i]^{(2)}$ for each corrupted server $S_i \in \mathcal{C}$. Instead, Sim_0 computes the secret of this sharing first and then randomly samples the whole sharing based on the shares for servers in $\mathcal{C}$ and the secret (in particular, during this process, when S_j is corrupted, the whole sharing $[\boldsymbol{y}_i]^{(2)}$ for each $S_i \in \mathcal{C}$ has been sampled). Then Sim_0 computes the whole

sharing $[\boldsymbol{o}_i]^{(2)}$ by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)} = [\boldsymbol{r}'_i]^{(2)}$ for each server $S_i \in \mathcal{C}$.

Since for each honest server S_i, $[\boldsymbol{r}'_i]^{(2)}$ is generated randomly by $\mathcal{F}_{\mathsf{prep}}$, where the shares for servers in $\mathcal{C}$ has already been generated, $[\boldsymbol{y}_i]^{(2)}$ is also completely random when these servers' shares are fixed. Therefore, we only change the order of generating S_j and the remaining honest servers' shares of each pair of $[\boldsymbol{r}'_i]^{(2)}$ and $[\boldsymbol{y}_i]^{(2)}$ without changing their distributions.

Similarly, for each corrupted server S_i, $[\boldsymbol{o}_i]^{(2)}$ is generated randomly by $\mathcal{F}_{\mathsf{prep}}$ based on the secret, $[\boldsymbol{y}_i]^{(2)}$ is also completely random when the shares for servers in $\mathcal{C}$ and the secret are fixed. Therefore, we only change the order of generating S_j and the remaining honest servers' shares of each pair of $[\boldsymbol{o}_i]^{(2)}$ and $[\boldsymbol{y}_i]^{(2)}$ without changing their distributions.

Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

Note that for each server S_i, the remaining honest servers' shares of $[\boldsymbol{o}_i]^{(2)}$ in these executions of $\Pi_{\mathsf{Transpose}}$ are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted. If S_i is honest, the remaining honest servers' shares of $[\boldsymbol{r}'_i]^{(2)}$ are also not used during the simulation, Sim_0 doesn't generate them in future hybrids as well until another server is corrupted.

$\mathbf{Hyb}_2$: In this hybrid, while **Handling Fan-out Gates** during the evaluation phase, for the first execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Tran}, Sim_0 doesn't follow the protocol to compute each server S_i's shares of $([\boldsymbol{x}_1^*], \ldots, [\boldsymbol{x}_k^*])$ by $\boldsymbol{y}_i - \boldsymbol{r}_i$. Instead, Sim_0 computes them by $F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})$. Since $[\boldsymbol{r}_i]^{(2)}$ is computed by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)}$, $\boldsymbol{y}_i - \boldsymbol{r}_i = F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})$ always holds. Therefore, we only change the way of generating each server S_i's shares of $([\boldsymbol{x}_1^*], \ldots, [\boldsymbol{x}_k^*])$ without changing their distributions. Thus, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ have the same output distribution.

Note that for each remaining honest server S_i, the secrets $\boldsymbol{r}_i$ in these executions of $\Pi_{\mathsf{Transpose}}$ are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted.

$\mathbf{Hyb}_3$: In this hybrid, while **Handling Fan-out Gates** during the evaluation phase, for the first execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Tran}, Sim_0 doesn't compute each server S_i's shares of $([\boldsymbol{x}_1^*], \ldots, [\boldsymbol{x}_k^*])$ by $F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})$. Instead, when a server S_j is newly corrupted, Sim_0 randomly generates the value of $\boldsymbol{y}_i = F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell}) + \boldsymbol{r}_i$ for $j = i$ and each corrupted server $S_i \in \mathcal{C}$, and then Sim_1 uses it to compute S_i's shares of $([\boldsymbol{x}_1^*], \ldots, [\boldsymbol{x}_k^*])$ by $\boldsymbol{y}_i - \boldsymbol{r}_i$ (during this process, when S_j is corrupted, the shares for the servers in $\mathcal{C}$ has been generated, so we only need to sample S_j's shares based on them). Then, Sim_0 samples the remaining honest servers' shares of $([\boldsymbol{x}_1^*], \ldots, [\boldsymbol{x}_k^*])$ based on the secrets and the corrupted parties' shares. Note that for each honest server S_i, the honest servers' shares and the secrets of the random sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ associated with this execution of $\Pi_{\mathsf{Transpose}}$ are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrutped. Sim_0 only generates it when the server is corrupted later.

Since $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell}$ are all sampled randomly, for each corrupted server S_i (including S_j), the computation of $F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})$ is just the generation process of S_i's shares of some random Σ-sharings based on the secrets. Thus, the corrupted servers' $F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})$ are random and independent of the secret $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$, so $F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell}) + \boldsymbol{r}_i$ for each corrupted S_i is also random. Therefore, we only change the order of generating each corrupted server S_i's $F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell}) + \boldsymbol{r}_i$ and $F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})$ without changing their distributions. Besides, since $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell}$ are all sampled randomly, the remaining honest servers' shares of $([\boldsymbol{x}_1^*], \ldots, [\boldsymbol{x}_k^*])$ are randomly sampled based on corrupted servers' shares and the secrets in both hybrids. Thus, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have the same output distribution.

Now, all the $\Sigma^{(2)}$-sharing reconstructed in this step are sampled independently. Sim_0 samples them at the beginning of the simulation.

$\mathbf{Hyb}_4$: In this hybrid, while **Handling Fan-out Gates** during the evaluation phase, for the second execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Tran}, when a server S_j is newly corrupted, Sim_0 doesn't follow the protocol to compute the remaining honest servers' shares of $[\boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) for each server S_i. Instead, Sim_0 randomly samples the whole sharing in the beginning. In addition, Sim_0 doesn't follow the protocol to compute the servers' shares of $([L_1(\boldsymbol{x}_1)], \ldots, [L_k(\boldsymbol{x}_k)])$. Instead, Sim_0 computes each corrupted server S_i's shares by $\boldsymbol{y}_i - \boldsymbol{r}_i$ and samples the honest servers' shares based on the secrets and the corrupted

servers' shares. In addition, Sim_0 does not follow the protocol to emulate $\mathcal{F}_{\mathsf{prep}}$ to generate S_j and the remaining honest servers' shares of the corresponding sharing $[\boldsymbol{o}_i]^{(2)}$. Instead, Sim_0 computes these servers' shares of $[\boldsymbol{o}_i]^{(2)}$ by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)} - [\boldsymbol{r}'_i]^{(2)}$ for S_j. Note that the secrets and the remaining honest servers' shares of $[\boldsymbol{o}_i]^{(2)}$ for each S_i, the remaining honest servers' shares of $[\boldsymbol{r}'_i]^{(2)}$ for each remaining honest S_i and the random sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ associated with each execution of $\Pi_{\mathsf{Transpose}}$ are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted. For the same reasons in $\mathbf{Hyb}_1$, $\mathbf{Hyb}_2$, and $\mathbf{Hyb}_3$, we conclude that $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ have the same output distribution.

$\mathbf{Hyb}_5$: In this hybrid, while **Permuting the Secrets** for the first time during the evaluation phase for each layer, for the first execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Tran}, when a server S_j is newly corrupted, Sim_0 doesn't follow the protocol to compute the remaining honest servers' shares of $[\boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) for each server S_i. Instead, Sim_0 randomly samples the whole sharing in the beginning. In addition, Sim_0 doesn't follow the protocol to compute the servers' shares of $([\boldsymbol{x}_1^*], \ldots, [\boldsymbol{x}_k^*])$. Instead, Sim_0 computes each corrupted server S_i's shares by $\boldsymbol{y}_i - \boldsymbol{r}_i$ and samples the honest servers' shares based on the secrets and the corrupted servers' shares. In addition, Sim_0 does not follow the protocol to emulate $\mathcal{F}_{\mathsf{prep}}$ to generate S_j and the remaining honest servers' shares of the corresponding sharing $[\boldsymbol{o}_i]^{(2)}$. Instead, Sim_0 computes these servers' shares of $[\boldsymbol{o}_i]^{(2)}$ by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)} - [\boldsymbol{r}'_i]^{(2)}$ for S_j. Note that the secrets and the remaining honest servers' shares of $[\boldsymbol{o}_i]^{(2)}$ for each S_i, the remaining honest servers' shares of $[\boldsymbol{r}'_i]^{(2)}$ for each remaining honest S_i and the random sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ associated with each execution of $\Pi_{\mathsf{Transpose}}$ are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted. For the same reasons in $\mathbf{Hyb}_1$, $\mathbf{Hyb}_2$, and $\mathbf{Hyb}_3$, we conclude that $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have the same output distribution.

$\mathbf{Hyb}_6$: In this hybrid, while **Permuting the Secrets** for the first time during the evaluation phase for each layer, for the second execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Tran}, when a server S_j is newly corrupted, Sim_0 doesn't follow the protocol to compute the remaining honest servers' shares of $[\boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) for each server S_i. Instead, Sim_0 randomly samples the whole sharing in the beginning. In addition, Sim_0 doesn't follow the protocol to compute the servers' shares of $([L_1(\boldsymbol{x}_1)], \ldots, [L_k(\boldsymbol{x}_k)])$. Instead, Sim_0 computes each corrupted server S_i's shares by $\boldsymbol{y}_i - \boldsymbol{r}_i$ and samples the honest servers' shares based on the secrets and the corrupted servers' shares. In addition, Sim_0 does not follow the protocol to emulate $\mathcal{F}_{\mathsf{prep}}$ to generate S_j and the remaining honest servers' shares of the corresponding sharing $[\boldsymbol{o}_i]^{(2)}$. Instead, Sim_0 computes these servers' shares of $[\boldsymbol{o}_i]^{(2)}$ by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)} - [\boldsymbol{r}'_i]^{(2)}$ for S_j. Note that the secrets and the remaining honest servers' shares of $[\boldsymbol{o}_i]^{(2)}$ for each S_i, the remaining honest servers' shares of $[\boldsymbol{r}'_i]^{(2)}$ for each remaining honest S_i and the random sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ associated with each execution of $\Pi_{\mathsf{Transpose}}$ are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted. For the same reasons in $\mathbf{Hyb}_1$, $\mathbf{Hyb}_2$, and $\mathbf{Hyb}_3$, we conclude that $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ have the same output distribution.

Note that the remaining honest parties' shares of $[\boldsymbol{s}]$ for each input Σ-sharing output by $\mathcal{F}_{\mathsf{input}}^{P_i}$ for an honest P_i are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted.

$\mathbf{Hyb}_7$: In this hybrid, while **Collecting Secrets from Previous Layers** during the evaluation phase, for the first execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Collect} while collecting secrets from previous layers in each layer, when a server S_j is newly corrupted, Sim_0 doesn't follow the protocol to compute the remaining honest servers' shares of $[\boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) for each server S_i. Instead, Sim_0 randomly samples the whole sharing in the beginning. In addition, Sim_0 doesn't follow the protocol to compute the servers' shares of $([\boldsymbol{y}_1^*], \ldots, [\boldsymbol{y}_k^*])$ (in Π_{Collect}). Instead, Sim_0 computes each corrupted server S_i's shares by $\boldsymbol{y}_i - \boldsymbol{r}_i$ and samples the honest servers' shares based on the secrets and the corrupted servers' shares. In addition, Sim_0 does not follow the protocol to emulate $\mathcal{F}_{\mathsf{prep}}$ to generate S_j and the remaining honest servers' shares of the corresponding sharing $[\boldsymbol{o}_i]^{(2)}$. Instead, Sim_0 computes these servers' shares of $[\boldsymbol{o}_i]^{(2)}$ by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)} - [\boldsymbol{r}'_i]^{(2)}$ for S_j. Note that the secrets and the remaining honest servers' shares of $[\boldsymbol{o}_i]^{(2)}$ for each S_i, the remaining honest servers' shares of $[\boldsymbol{r}'_i]^{(2)}$ for each remaining honest S_i and the random sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ associated with each execution of $\Pi_{\mathsf{Transpose}}$ are not used in the later

simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted. For the same reasons in $\mathbf{Hyb}_1$, $\mathbf{Hyb}_2$, and $\mathbf{Hyb}_3$, we conclude that $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have the same output distribution.

$\mathbf{Hyb}_8$: In this hybrid, while **Collecting Secrets from Previous Layers** during the evaluation phase, for the second execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Collect} while collecting secrets from previous layers in each layer, when a server S_j is newly corrupted, Sim_0 doesn't follow the protocol to compute the remaining honest servers' shares of $[\boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) for each server S_i. Instead, Sim_0 randomly samples the whole sharing in the beginning. In addition, Sim_0 doesn't follow the protocol to compute the servers' shares of $([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_k])$ (in Π_{Collect}). Instead, Sim_0 computes each corrupted server S_i's shares by $\boldsymbol{y}_i - \boldsymbol{r}_i$ and samples the honest servers' shares based on the secrets and the corrupted servers' shares. In addition, Sim_0 does not follow the protocol to emulate $\mathcal{F}_{\mathsf{prep}}$ to generate S_j and the remaining honest servers' shares of the corresponding sharing $[\boldsymbol{o}_i]^{(2)}$. Instead, Sim_0 computes these servers' shares of $[\boldsymbol{o}_i]^{(2)}$ by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)} - [\boldsymbol{r}'_i]^{(2)}$ for S_j. Note that the secrets and the remaining honest servers' shares of $[\boldsymbol{o}_i]^{(2)}$ for each S_i, the remaining honest servers' shares of $[\boldsymbol{r}'_i]^{(2)}$ for each remaining honest S_i and the random sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ associated with each execution of $\Pi_{\mathsf{Transpose}}$ are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted. For the same reasons in $\mathbf{Hyb}_1$, $\mathbf{Hyb}_2$, and $\mathbf{Hyb}_3$, we conclude that $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ have the same output distribution.

Note that the remaining honest servers' shares of the sharings $([\boldsymbol{y}_1^*], \ldots, [\boldsymbol{y}_k^*])$, $([\boldsymbol{y}_1^*]^{(2)}, \ldots, [\boldsymbol{y}_k^*]^{(2)})$, and $([\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_k]^{(2)})$ in each execution of Π_{Collect} while collecting secrets from previous layers are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted.

$\mathbf{Hyb}_9$: In this hybrid, while **Permuting the Secrets** for the second time during the evaluation phase for each layer, for the first execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Tran}, when a server S_j is newly corrupted, Sim_0 doesn't follow the protocol to compute the remaining honest servers' shares of $[\boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) for each server S_i. Instead, Sim_0 randomly samples the whole sharing in the beginning. In addition, Sim_0 doesn't follow the protocol to compute the servers' shares of $([\boldsymbol{x}_1^*], \ldots, [\boldsymbol{x}_k^*])$. Instead, Sim_0 computes each corrupted server S_i's shares by $\boldsymbol{y}_i - \boldsymbol{r}_i$ and samples the honest servers' shares based on the secrets and the corrupted servers' shares. In addition, Sim_0 does not follow the protocol to emulate $\mathcal{F}_{\mathsf{prep}}$ to generate S_j and the remaining honest servers' shares of the corresponding sharing $[\boldsymbol{o}_i]^{(2)}$. Instead, Sim_0 computes these servers' shares of $[\boldsymbol{o}_i]^{(2)}$ by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)} - [\boldsymbol{r}'_i]^{(2)}$ for S_j. Note that the secrets and the remaining honest servers' shares of $[\boldsymbol{o}_i]^{(2)}$ for each S_i, the remaining honest servers' shares of $[\boldsymbol{r}'_i]^{(2)}$ for each remaining honest S_i and the random sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ associated with each execution of $\Pi_{\mathsf{Transpose}}$ are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted. For the same reasons in $\mathbf{Hyb}_1$, $\mathbf{Hyb}_2$, and $\mathbf{Hyb}_3$, we conclude that $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ have the same output distribution.

$\mathbf{Hyb}_{10}$: In this hybrid, while **Permuting the Secrets** for the second time during the evaluation phase for each layer, for the second execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Tran}, when a server S_j is newly corrupted, Sim_0 doesn't follow the protocol to compute the remaining honest servers' shares of $[\boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) for each server S_i. Instead, Sim_0 randomly samples the whole sharing in the beginning. In addition, Sim_0 doesn't follow the protocol to compute the servers' shares of $([L_1(\boldsymbol{x}_1)], \ldots, [L_k(\boldsymbol{x}_k)])$. Instead, Sim_0 computes each corrupted server S_i's shares by $\boldsymbol{y}_i - \boldsymbol{r}_i$ and samples the honest servers' shares based on the secrets and the corrupted servers' shares. In addition, Sim_0 does not follow the protocol to emulate $\mathcal{F}_{\mathsf{prep}}$ to generate S_j and the remaining honest servers' shares of the corresponding sharing $[\boldsymbol{o}_i]^{(2)}$. Instead, Sim_0 computes these servers' shares of $[\boldsymbol{o}_i]^{(2)}$ by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)} - [\boldsymbol{r}'_i]^{(2)}$ for S_j. Note that the secrets and the remaining honest servers' shares of $[\boldsymbol{o}_i]^{(2)}$ for each S_i, the remaining honest servers' shares of $[\boldsymbol{r}'_i]^{(2)}$ for each remaining honest S_i and the random sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ associated with each execution of $\Pi_{\mathsf{Transpose}}$ are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted. For the same reasons in $\mathbf{Hyb}_1$, $\mathbf{Hyb}_2$, and $\mathbf{Hyb}_3$, we conclude that $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ have the same output distribution.

$\mathbf{Hyb}_{11}$: In this hybrid, while **Evaluating Multiplication Gates and Addition Gates** during the

evaluation phase, for the first execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Multi} while evaluating the multiplication gates in each layer, when a server S_j is newly corrupted, Sim_0 doesn't follow the protocol to compute the remaining honest servers' shares of $[\boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) for each server S_i. Instead, Sim_0 randomly samples the whole sharing in the beginning. In addition, Sim_0 doesn't follow the protocol to compute the servers' shares of $([\boldsymbol{z}_1^*], \ldots, [\boldsymbol{z}_k^*])$. Instead, Sim_0 computes each corrupted server S_i's shares by $\boldsymbol{y}_i - \boldsymbol{r}_i$ and samples the honest servers' shares based on the secrets and the corrupted servers' shares. In addition, Sim_0 does not follow the protocol to emulate $\mathcal{F}_{\mathsf{prep}}$ to generate S_j and the remaining honest servers' shares of the corresponding sharing $[\boldsymbol{o}_i]^{(2)}$. Instead, Sim_0 computes these servers' shares of $[\boldsymbol{o}_i]^{(2)}$ by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)} - [\boldsymbol{r}'_i]^{(2)}$ for S_j. Note that the secrets and the remaining honest servers' shares of $[\boldsymbol{o}_i]^{(2)}$ for each S_i, the remaining honest servers' shares of $[\boldsymbol{r}'_i]^{(2)}$ for each remaining honest S_i and the random sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ associated with each execution of $\Pi_{\mathsf{Transpose}}$ are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted. For the same reasons in $\mathbf{Hyb}_1$, $\mathbf{Hyb}_2$, and $\mathbf{Hyb}_3$, we conclude that $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ have the same output distribution.

$\mathbf{Hyb}_{12}$: In this hybrid, while **Evaluating Multiplication Gates and Addition Gates** during the evaluation phase, for the second execution of $\Pi_{\mathsf{Transpose}}$ in each execution of Π_{Multi} while evaluating the multiplication gates in each layer, when a server S_j is newly corrupted, Sim_0 doesn't follow the protocol to compute the remaining honest servers' shares of $[\boldsymbol{y}_i]^{(2)}$ (in $\Pi_{\mathsf{Transpose}}$) for each server S_i. Instead, Sim_0 randomly samples the whole sharing in the beginning. In addition, Sim_0 doesn't follow the protocol to compute the servers' shares of $([\boldsymbol{z}_1], \ldots, [\boldsymbol{z}_k])$. Instead, Sim_0 computes each corrupted server S_i's shares by $\boldsymbol{y}_i - \boldsymbol{r}_i$ and samples the honest servers' shares based on the secrets and the corrupted servers' shares. In addition, Sim_0 does not follow the protocol to emulate $\mathcal{F}_{\mathsf{prep}}$ to generate S_j and the remaining honest servers' shares of the corresponding sharing $[\boldsymbol{o}_i]^{(2)}$. Instead, Sim_0 computes these servers' shares of $[\boldsymbol{o}_i]^{(2)}$ by $[\boldsymbol{y}_i]^{(2)} - [F_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n\ell})]^{(2)} - [\boldsymbol{r}'_i]^{(2)}$ for S_j. Note that the secrets and the remaining honest servers' shares of $[\boldsymbol{o}_i]^{(2)}$ for each S_i, the remaining honest servers' shares of $[\boldsymbol{r}'_i]^{(2)}$ for each remaining honest S_i and the random sharings $[\boldsymbol{u}_1]^{(2)}, \ldots, [\boldsymbol{u}_{n\ell}]^{(2)}$ associated with each execution of $\Pi_{\mathsf{Transpose}}$ are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted. For the same reasons in $\mathbf{Hyb}_1$, $\mathbf{Hyb}_2$, and $\mathbf{Hyb}_3$, we conclude that $\mathbf{Hyb}_{12}$ and $\mathbf{Hyb}_{11}$ have the same output distribution.

Note that the remaining honest servers' shares of the sharings $([\boldsymbol{z}_1^*], \ldots, [\boldsymbol{z}_k^*])$, $([\boldsymbol{z}_1^*]^{(2)}, \ldots, [\boldsymbol{z}_k^*]^{(2)})$, and $([\boldsymbol{z}_1]^{(2)}, \ldots, [\boldsymbol{z}_k]^{(2)})$ in each execution of Π_{Multi} while evaluating the multiplication gates are not used in the later simulation if no servers are corrupted afterwards, Sim_0 doesn't generate them in future hybrids until another server is corrupted.

$\mathbf{Hyb}_{13}$: In this hybrid, for each batch of output wires attached to an honest client C_i, Sim_0 doesn't follow the protocol to compute the output sharing for this batch of output. Instead, when a server S_j is newly corrupted, Sim_0 randomly samples S_j's shares of the sharing based on the shares for the servers in $\mathcal{C}$. Since the total number of corrupted parties is smaller than t, the sampling of S_j's share of the sharing is independent of the secret. Thus, we only change the order of sampling different servers' shares without changing their distributions. Thus, $\mathbf{Hyb}_{13}$ and $\mathbf{Hyb}_{12}$ have the same output distribution.

$\mathbf{Hyb}_{14}$: In this hybrid, for each batch of output wires attached to a corrupted client, Sim_0 doesn't follow the protocol to generate the $\Sigma^{(2)}$-sharing with an all-zero secret attached to this batch of output and compute the output sharing. Instead, Sim_0 directly sample the output sharing based on the output values computed from the circuit C'. Then, Sim_0 computes the $\Sigma^{(2)}$-sharing with an all-zero secret based on the output sharing. Note that the output sharing is masked by the $\Sigma^{(2)}$-sharing with an all-zero secret that is completely random based on its secret, so is the output sharing. Therefore, we only change the order of generating the two sharings without changing their distributions. Thus, $\mathbf{Hyb}_{14}$ and $\mathbf{Hyb}_{13}$ have the same output distribution.

$\mathbf{Hyb}_{15}$: In this hybrid, Sim_0 does not compute the corrupted clients' outputs following the circuit C'. Instead, Sim_0 directly obtains the outputs from $\mathcal{F}$. In addition, honest clients don't compute their output by themselves. Instead, honest clients get their output from $\mathcal{F}$. Sim_0 doesn't generate the remaining honest servers' shares of the Σ-sharings for intermediate wire values until another server is corrupted. These shares

are not used in the simulation, so this doesn't affect the output distribution. Since the computation process of $\mathcal{F}$ of $\boldsymbol{y}$ of each client (either honest or corrupted) is the computation of circuit C, which has the same output as C', and the computation process of the output sharings by the servers is just the computation of C' in a secret-shared form. Thus, we only change the way to generate the outputs without changing their distributions. Thus, $\mathbf{Hyb}_{15}$ and $\mathbf{Hyb}_{14}$ have the same output distribution.

Note that $\mathbf{Hyb}_{15}$ is the ideal-world scenario, Π_0 satisfies the requirements listed in Section 4.

D.5 Towards Underlying Protocol for Malicious 2PC

The underlying protocol Π_0' is only slightly modified from Π_0 from two aspects:

1. The corruption threshold is set to be $t = n/6$ such that a $\Sigma^{(3)}$-sharings has error-correction against up to t incorrect shares.

2. We allow the adversary $\mathcal{A}$ choose the randomness for $\mathcal{F}_{\mathsf{input}}^{P_i}$ if P_i is corrupted. The corrupted clients must be chosen at the beginning of the protocol.

These changes only incur a constant overhead over the cost of Π_0, and they won't affect the security of Π_0. Thus, we can still follow the steps of Sim_0 to build Sim_0', except that the outputs of $\mathcal{F}_{\mathsf{input}}^{P_i}$ for a corrupted client P_i are no longer sampled by Sim_0'. Instead, Sim_0' receives the randomness from $\mathcal{A}$ and then generates the outputs based on the randomness.

E Instantiation of the Linear Secret Sharing Scheme

The linear secret sharing scheme Σ we used is based on the secret sharing scheme Σ' given in [CC06] from algebraic geometry code. We first give a brief introduction to basic algebraic geometry, which is borrowed from [CC06, GLOS25].

E.1 Basic Algebraic Geometry

Let C be a smooth, projective, absolutely irreducible curve defined over $\mathbb{F}_q$, and let g denote the genus of C. Let $\overline{\mathbb{F}_q}$ denote the algebraic closure of $\mathbb{F}_q$. A plane such curve can be represented by some polynomial $F(X,Y) \in \mathbb{F}_q[X,Y]$ that is irreducible in $\overline{\mathbb{F}_q}[X,Y]$.

The affine part of the curve is defined as the set of points $P \in \overline{\mathbb{F}_q}^2$ such that $F(P) = 0$. By taking its projective closure, which amounts to introducing an extra variable, homogenizing the polynomial, and considering the zeroes in the two-dimensional projective space $\mathbb{P}^2(\overline{\mathbb{F}_q})$, one obtains the entire curve. More generally, curves defined over $\mathbb{F}_q$ is the "set of zeroes" in $\mathbb{P}^m(\overline{\mathbb{F}_q})$ of a homogeneous ideal $I \subset \mathbb{F}_q[X_0, \ldots, X_m]$, where I is such that its function field has transcendence degree 1 over the ground field, i.e., it is a one-dimensional variety. For a projective curve $V \subset \mathbb{P}^m(\overline{\mathbb{F}_q})$ defined by $m-1$ irreducible functions $f_1, \ldots, f_{m-1}$, if the at every point $P \in \overline{\mathbb{F}_q}^m$ in the affine part of V (such that $f_1(P) = \cdots = f_{m-1}(P) = 1$) is nonsingular, i.e. the $(m-1) \times m$ matrix containing the m partial derivatives of $f_1, \ldots, f_{m-1}$ has rank $m-1$, V is said to be smooth.

$\overline{\mathbb{F}_q}(C)$ denotes the function field of the curve. Very briefly, it consists of all fractions of polynomials $a, b \in \overline{\mathbb{F}_q}[X_0, \ldots, X_m]$, $b \notin I$, such that both are homogeneous of the same degree, under the equivalence relation that $a/b \sim a'/b'$ if $ab' \equiv a'b \bmod I$. The elements can be viewed as maps from the curve to $\overline{\mathbb{F}_q}$, and they have at most a finite number of poles and zeroes unless it is the zero function. Their "multiplicities add up to zero."

Since C is smooth at each point $P \in C$ by assumption, the local ring $\mathcal{O}_P(C)$ of functions $f \in \overline{\mathbb{F}_q}(C)$ that are well-defined at P (equivalently, the ones that do not have a pole at P) is a discrete valuation ring. Thus, at each $P \in C$, there exists $t \in \overline{\mathbb{F}_q}(C)$ (a uniformizing parameter) such that $t(P) = 0$ and each $f \in \mathcal{O}_P(C)$ can be uniquely written as $f = u \cdot t^{\nu_P(f)}$. Here, $u \in \mathcal{O}_P(C)$ is a unit (i.e. $u(P) \neq 0$), and $\nu_P(f)$ is a

non-negative integer. This valuation ν_P extends to all of $\overline{\mathbb{F}_q}(C)$, by defining $\nu_P(f) = -\nu_P(1/f)$ if f has a pole at P.

A divisor is a formal sum $\sum_{P\in C} m_p \cdot (P)$ with integer coefficients m_p taken over all points P of the curve C. Divisors are required to have finite support, i.e., they are zero except possibly at finitely many points. The divisor of $f \in \mathbb{F}_q(C)$ is defined as $\mathsf{div}(f) = \sum_{P\in C} \nu_P(f) \cdot (P)$. It holds that $\mathsf{deg}(\mathsf{div}(f)) = 0$. The degree $\mathsf{deg}(D)$ of a divisor D is the sum $\sum_{P\in C} m_P \in \mathbb{Z}$ of its coefficients m_P.

The Riemann-Roch space associated with a divisor D is defined as $\mathcal{L}(D) = \{f \in \overline{\mathbb{F}_q}(C) | \mathsf{div}(f) + D \geq 0\} \cup \{0\}$. This is an $\overline{\mathbb{F}_q}$-vector space. The (partial) ordering "$\geq$" refers to the comparison of integer vectors and declaring one larger than the other if this holds coordinate-wise. Its dimension is denoted $\ell(D)$. This dimension is equal to 0 if $\mathsf{deg}(D) < 0$. The Riemann-Roch Theorem is concerned with the dimensions of those spaces. It says that $\ell(D) - \ell(K - D) = \mathsf{deg}(D) + 1 - g$. Here K is a canonical divisor. These are the divisors K of degree $2g - 2$ and $\ell(K) = g$. It follows immediately that $\ell(D) = \mathsf{deg}(D) + 1 - g$ if $\mathsf{deg}(D)$ is at least $2g - 1$.

An $\mathbb{F}_q$-rational point on C is one whose projective coordinates can be chosen in $\mathbb{F}_q$. The set of $\mathbb{F}_q$-rational point on C is denoted by $C(\mathbb{F}_q)$. Below is a basic lemma in algebraic geometry.

Lemma 3 ([CC06]). *Let E be a divisor on a smooth, projective, absolutely irreducible curve C that is defined over $\mathbb{F}_q$, and suppose that $\ell(E) > 0$. Then each $f \in \mathcal{L}(E)$ is uniquely determined by evaluations of f on any $\mathsf{deg}(E) + 1$ $\mathbb{F}_q$-rational points on C outside the support of E.*

E.2 The Linear Secret Sharing Scheme

In the linear secret sharing scheme Σ' in [CC06], n parties agree on a smooth projective absolutely irreducible curve C with genus g defined over $\mathbb{F}_q$ and distinct $\mathbb{F}_q$-rational points

$$Q, P_{-1}, \ldots, P_{-k}, P_1, \ldots, P_n \in C(\mathbb{F}_q).$$

For a divisor D defined by $D = (2g + t + k - 1) \cdot (Q)$, the sharing algorithm randomly selects $f \in \mathcal{L}(D)$ subject to

$$(f(P_{-1}), \ldots, f(P_{-k})) = \boldsymbol{s}.$$

Then the Σ' secret sharing is defined by

$$[\boldsymbol{s}]' = \Sigma'.\mathsf{Sh}(\boldsymbol{s}, r) = (f(P_1), \ldots, f(P_n)) \in \mathbb{F}_q^n.$$

Lemma 3 implies that the secret $\boldsymbol{s}$ of $[\boldsymbol{s}]$ can be reconstructed from any $\mathsf{deg}(D)+1$ parties' shares, which implies the reconstruction algorithm $\Sigma'.\mathsf{Rec}$. In [CC06], the authors have shown that Σ' is an $(n, t, k, 1)$-LSSS over $\mathbb{F}_q$.

Defining Σ. If $q = 2^\ell$ (ℓ will be specified later), we can restrict the secret on $\mathbb{F}_2^k$ where each entry of the secret is stored in a subspace of $\mathbb{F}_q$ that is isomorphic to $\mathbb{F}_2$. Besides, each party's share in $\mathbb{F}_q = \mathbb{F}_{2^\ell}$ can be written as a vector in over $\mathbb{F}_2^\ell$. In this way, we obtain an (n, t, k, ℓ)-LSSS over $\mathbb{F}_2$. We take such a sharing to be Σ.

Formally, there exists a bijective $\mathbb{F}_2$-linear map $\mathsf{Conv} : \mathbb{F}_2^\ell \to \mathbb{F}_q$, and both $\mathsf{Conv}, \mathsf{Conv}^{-1}$ can be efficiently computed. In the following, for $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in (\mathbb{F}_2^\ell)^n$, we use $\mathsf{Conv}(\boldsymbol{x})$ to denote $(\mathsf{Conv}(\boldsymbol{x}_1), \ldots, \mathsf{Conv}(\boldsymbol{x}_n))$. And for $\boldsymbol{y} = (y_1, \ldots, y_n) \in \mathbb{F}_q^n$, we use $\mathsf{Conv}^{-1}(\boldsymbol{y})$ to denote $(\mathsf{Conv}^{-1}(y_1), \ldots, \mathsf{Conv}^{-1}(y_n))$.

We define Σ as follows:

- For a vector of secrets $\boldsymbol{s} \in \mathbb{F}_2^k$,

$$\Sigma.\mathsf{Sh}(\boldsymbol{s}, r) = (\mathsf{Conv}^{-1}(\Sigma'.\mathsf{Sh}_1(\boldsymbol{s}, \mathsf{Conv}(r))), \ldots, \mathsf{Conv}^{-1}(\Sigma'.\mathsf{Sh}_n(\boldsymbol{s}, \mathsf{Conv}(r)))).$$

- For a Σ-sharing $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n)$,

$$\Sigma.\mathsf{Rec}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) = \Sigma'.\mathsf{Rec}(\mathsf{Conv}(\boldsymbol{c}_1), \ldots, \mathsf{Conv}(\boldsymbol{c}_n)).$$

Defining $\Sigma^{(2)}$. We follow [GLOS25] to define $\Sigma^{(2)}$. We first define an LSSS $\Sigma^{(2)'}$ over $\mathbb{F}_q$, denoted by $[\cdot]^{(2)'}$. $\Sigma^{(2)'}$ is the same as Σ' except that the divisor D is replaced by $2D$ (i.e. f is selected from $\mathcal{L}(2D)$). Note that for any $f_1, f_2 \in \mathcal{L}(D)$, it holds that:

$$(\mathsf{div}(f_1) + D) + (\mathsf{div}(f_2) + D) = \mathsf{div}(f_1 \cdot f_2) + 2D,$$

which implies that $f_1 \cdot f_2 \in \mathcal{L}(2D)$. Thus, the multiplication of two sharings $[\boldsymbol{s}_1]' \cdot [\boldsymbol{s}_2]'$ is a $\Sigma^{(2)'}$-sharing $[\boldsymbol{s}_1 * \boldsymbol{s}_2]^{(2)'}$. Note that for any $x^{(1)}, x^{(2)} \in \mathbb{F}_q$ with $\mathsf{Conv}^{-1}(x^{(1)}) = (x_1^{(1)}, \ldots, x_\ell^{(1)}), \mathsf{Conv}^{-1}(x^{(2)}) = (x_1^{(2)}, \ldots, x_\ell^{(2)}) \in \mathbb{F}_2^\ell$, their product $x^{(1)} \cdot x^{(2)}$ is uniquely determined by $(x_1^{(1)}, \ldots, x_\ell^{(1)}) \otimes (x_1^{(2)}, \ldots, x_\ell^{(2)}) \in \mathbb{F}_2^{\ell^2}$. Let $\mathsf{Conv}_2 : \mathbb{F}_2^{\ell^2} \to \mathbb{F}_q$ be a $\mathbb{F}_2$-linear function that maps $\mathsf{Conv}^{-1}(x^{(1)}) \otimes \mathsf{Conv}^{-1}(x^{(2)})$ to $x^{(1)} \cdot x^{(2)}$. Let Conv_2^{-1} be a randomized $\mathbb{F}_2$-linear function such that for all $x \in \mathbb{F}_q$, $\mathsf{Conv}_2^{-1}(x)$ outputs a random vector $\boldsymbol{y} \in \mathbb{F}_2^{\ell^2}$ such that $\mathsf{Conv}_2(\boldsymbol{y}) = x$.

We define $\Sigma^{(2)}$ as follows:

- For a vector of secrets $\boldsymbol{s} \in \mathbb{F}_2^k$,

$$\begin{aligned}&\Sigma^{(2)}.\mathsf{Sh}(\boldsymbol{s}, r \| r_1 \| \ldots \| r_n)\\ =&(\mathsf{Conv}_2^{-1}(\Sigma^{(2)'}.\mathsf{Sh}_1(\boldsymbol{s}, \mathsf{Conv}(r)), r_1), \ldots, \mathsf{Conv}_2^{-1}(\Sigma^{(2)'}.\mathsf{Sh}_n(\boldsymbol{s}, \mathsf{Conv}(r)), r_n)).\end{aligned}$$

- For a $\Sigma^{(2)}$-sharing $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n)$,

$$\Sigma^{(2)}.\mathsf{Rec}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) = \Sigma^{(2)'}.\mathsf{Rec}(\mathsf{Conv}_2(\boldsymbol{c}_1), \ldots, \mathsf{Conv}_2(\boldsymbol{c}_n)).$$

Defining $\Sigma^{(3)}$. Similarly, we may define $\Sigma^{(3)'}$ over $\mathbb{F}_q$ by replacing D by $3D$ in Σ'. For any $x^{(1)}, x^{(2)}, x^{(3)} \in \mathbb{F}_q$ with $\mathsf{Conv}^{-1}(x^{(1)}) = (x_1^{(1)}, \ldots, x_\ell^{(1)}), \mathsf{Conv}^{-1}(x^{(2)}) = (x_1^{(2)}, \ldots, x_\ell^{(2)}) \in \mathbb{F}_2^\ell, \mathsf{Conv}^{-1}(x^{(3)}) = (x_1^{(3)}, \ldots, x_\ell^{(3)}) \in \mathbb{F}_2^\ell$, their product $x^{(1)} \cdot x^{(2)} \cdot x^{(3)}$ is uniquely determined by $(x_1^{(1)}, \ldots, x_\ell^{(1)}) \otimes (x_1^{(2)}, \ldots, x_\ell^{(2)}) \otimes (x_1^{(3)}, \ldots, x_\ell^{(3)}) \in \mathbb{F}_2^{\ell^3}$. Let $\mathsf{Conv}_3 : \mathbb{F}_2^{\ell^3} \to \mathbb{F}_q$ be a $\mathbb{F}_2$-linear function that maps $\mathsf{Conv}^{-1}(x^{(1)}) \otimes \mathsf{Conv}^{-1}(x^{(2)}) \otimes \mathsf{Conv}^{-1}(x^{(3)})$ to $x^{(1)} \cdot x^{(2)} \cdot x^{(3)}$. Let Conv_3^{-1} be a randomized $\mathbb{F}_2$-linear function such that for all $x \in \mathbb{F}_q$, $\mathsf{Conv}_3^{-1}(x)$ outputs a random vector $\boldsymbol{y} \in \mathbb{F}_2^{\ell^3}$ such that $\mathsf{Conv}_3(\boldsymbol{y}) = x$.

We define $\Sigma^{(3)}$ as follows:

- For a vector of secrets $\boldsymbol{s} \in \mathbb{F}_2^k$,

$$\begin{aligned}&\Sigma^{(3)}.\mathsf{Sh}(\boldsymbol{s}, r \| r_1 \| \ldots \| r_n)\\ =&(\mathsf{Conv}_3^{-1}(\Sigma^{(3)'}.\mathsf{Sh}_1(\boldsymbol{s}, \mathsf{Conv}(r)), r_1), \ldots, \mathsf{Conv}_3^{-1}(\Sigma^{(3)'}.\mathsf{Sh}_n(\boldsymbol{s}, \mathsf{Conv}(r)), r_n)).\end{aligned}$$

- For a $\Sigma^{(3)}$-sharing $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n)$,

$$\Sigma^{(3)}.\mathsf{Rec}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) = \Sigma^{(3)'}.\mathsf{Rec}(\mathsf{Conv}_3(\boldsymbol{c}_1), \ldots, \mathsf{Conv}_3(\boldsymbol{c}_n)).$$

Parameter Choices. Let q be a square, there exists a family of curves [GS96] $\{C^{(j)}\}_{j=1}^\infty$ such that

$$\#C^{(j)}(\mathbb{F}_q) \geq (q - \sqrt{q}) \cdot \sqrt{q}^{j-1} \quad \text{and} \quad g(C^{(j)}) \leq \sqrt{q}^j.$$

Lemma 3 implies that the secret $\boldsymbol{s}$ of $[\boldsymbol{s}]$ can be reconstructed from any $\mathsf{deg}(D) + 1$ parties' shares. Since for any $f_1, f_2 \in \mathcal{L}(D)$, it holds that:

$$(\mathsf{div}(f_1) + D) + (\mathsf{div}(f_2) + D) = \mathsf{div}(f_1 \cdot f_2) + 2D.$$

Hence $f_1 \cdot f_2 \in \mathcal{L}(2D)$, so we need $3t + 6g + 3k - 3 < n$ to let Σ have 3-multiplicative reconstruction. Thus, we only need

$$3t + 4k + 6\sqrt{q}^j < (q - \sqrt{q}) \cdot \sqrt{q}^{j-1}$$

for some $k = \epsilon n$ (ϵ is a constant), i.e.

$$t < \left(\frac{1}{3} - \frac{3}{2(\sqrt{q}-1)}\right) \cdot (1-4\epsilon)n.$$

Take $t = n/4, q = 2^{10}, \epsilon = 1/40$, we get an $(n, n/4, n/40, 10)$-LSSS Σ.

Theorem 5 ([CC06]). *There exists an $(n, n/4, n/40, 10)$-LSSS Σ over $\mathbb{F}_2$ with $k = n/40 = \Theta(n)$ that has 3-multiplicative reconstruction.*

E.3 Error Correction

The error correction of the above LSSS can be achieved using a decoding procedure in the spirit of the Berlekamp-Welch algorithm [WB86]. Note that the error correction of Σ is equivalent to the error correction of Σ' since there is a bijective map between the shares of the two LSSSs, so we only focus on the error correction of Σ.

Suppose that during reconstruction, we receive a vector of shares

$$(y_1, \ldots, y_n),$$

where up to e shares may be corrupted by a malicious adversary. Recall that in a correct sharing, the shares are evaluations of a function $f \in \mathcal{L}(D)$ at the points $P_1, \ldots, P_n$, i.e., $y_i = f(P_i)$ for all honest parties.

To correct errors, we introduce an error-locator function $E \in \mathcal{L}(H)$, where H is a divisor whose support is disjoint from $\{P_1, \ldots, P_n\}$ and satisfies $\mathsf{deg}(H) = e$. We then search for functions

$$E \in \mathcal{L}(H), \qquad Q \in \mathcal{L}(D+H)$$

such that

$$Q(P_i) = y_i E(P_i) \qquad \text{for all } i \in [n].$$

Intuitively, the zeros of E correspond to the positions of corrupted shares. For each honest party we have $y_i = f(P_i)$, which implies

$$Q(P_i) = f(P_i)E(P_i).$$

Thus, the functions $Q = f \cdot E$ and E satisfy the above equations for all honest positions. The above constraints form a linear system over $\mathbb{F}_q$ in the coefficients of Q and E, which can be solved efficiently.

More concretely, we need the number of honest shares, i.e. $n-e$, to be at least $\mathsf{deg}(D+H)+1$. Then, by Lemma 3, the function $Q - fE$ must be identically zero, which implies $Q = fE$. Consequently, the recovered function $f = Q/E$ is exactly the original sharing function.

Note that we actually require $\Sigma^{(3)}$ to have error correction over t errors in our maliciously secure 2PC protocol. We require $n - t > \mathsf{deg}(3D+E) = 4t + 6g + 3k - 3$, i.e. $5t + 6g + 3k - 3 < n$. Thus, we only need

$$5t + 4k + 6\sqrt{q}^j < (q - \sqrt{q}) \cdot \sqrt{q}^{j-1}$$

for some $k = \epsilon' n$ (ϵ' is a constant), i.e.

$$t < \left(\frac{1}{5} - \frac{3}{2(\sqrt{q}-1)}\right) \cdot (1-4\epsilon)n.$$

We can take $n = 6t, q = 2^{12}, \epsilon = 1/40$ to meet the requirement.

F Security Proof for the Garbling Scheme

The correctness follows directly from the descriptions of the algorithms and the correctness of Π_0 when P, P' are all honest. We now prove the privacy, obliviousness, and authenticity of our garbling scheme.

F.1 Proof of Privacy

We prove the privacy by explicitly constructing the simulator $\mathsf{Sim}_{\mathsf{pri}}$. Then, we will show that the distributions of $(\tilde{C}, X, d)$ generated by the real garbling scheme and $\mathsf{Sim}_{\mathsf{pri}}$ are indistinguishable using hybrid arguments.

Simulator $\mathsf{Sim}_{\mathsf{pri}}$

$\mathsf{Sim}_{\mathsf{pri}}$ takes $(1^\kappa, C, C(x))$ as inputs and does the following:

1. **Initializing.** $\mathsf{Sim}_{\mathsf{pri}}$ follows the initializing step of Garble to initialize the underlying protocol Π_0, where the adversary of Π_0 corrupts P. $\mathsf{Sim}_{\mathsf{pri}}$ initializes $\mathcal{C} = \emptyset$ and $\mathsf{Corr}_j = 0$ for each $j = 1, \ldots, n$.
2. **Generating Output Mask.** $\mathsf{Sim}_{\mathsf{pri}}$ follows the algorithm description of Garble to sample $\mathsf{mask}, \boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$. $\mathsf{Sim}_{\mathsf{pri}}$ then runs Sim_0 to obtain P's view. In particular, while emulating $\mathcal{F}$ when running Sim_0, $\mathsf{Sim}_{\mathsf{pri}}$ sends $\mathsf{En}(C(x) \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}))$ to Sim_0 as $\mathcal{F}$'s output.
3. **Generating Output Labels.** For $i = 1, \ldots, \mathsf{rec} + n_o$:
   (a) $\mathsf{Sim}_{\mathsf{pri}}$ gets $\boldsymbol{s}_i$ from Sim_0. $\mathsf{Sim}_{\mathsf{pri}}$ then samples k random $\log 40T$-bit strings as $\boldsymbol{r}_{s_{i,\beta},\beta} = (r^{(1)}_{s_{i,\beta},\beta}, \ldots, r^{(\log 40T)}_{s_{i,\beta},\beta})$ for $\beta = 1, \ldots, k$ (where $s_{i,\beta}$ denotes the β-th bit of $\boldsymbol{s}_i$). Let $\boldsymbol{r}^{(\alpha)}_{\boldsymbol{s}_i} = (r^{(\alpha)}_{s_{i,1},1}, \ldots, r^{(\alpha)}_{s_{i,k},k})$ for each $\alpha = 1, \ldots, \log 40T$.
   (b) For $\alpha = 1, \ldots, \log 40T$, $\mathsf{Sim}_{\mathsf{pri}}$ randomly samples each sharing $[\boldsymbol{r}^{(\alpha)}_{\boldsymbol{s}_i}]^{(3)}$ based on the secret $\boldsymbol{r}^{(\alpha)}_{\boldsymbol{s}_i}$.
   (c) For each server S_j and each $a = 1, \ldots, \ell^2$, let $s^{S_j}_{i,a}$ be the a-th bit of S_j's share of $[\boldsymbol{s}_i]^{(2)}$ and
   $$Y^{S_j}_{(i-1)\ell^2+a, s^{S_j}_{i,a}} = \big(([\boldsymbol{r}^{(1)}_{\boldsymbol{s}_i}]^{(3)})^{S_j}_{[a\ell+1,(a+1)\ell]}, \ldots, ([\boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}_i}]^{(3)})^{S_j}_{[a\ell+1,(a+1)\ell]}\big).$$
4. **Sampling Wire Labels.** For each $j = 1, \ldots, n$ and each wire w of $\mathtt{Circ}^{S_j}$, $\mathsf{Sim}_{\mathsf{pri}}$ samples a random bit as $v_w \oplus \lambda_w$ and $\log 40T$-bit string as $k_{w,v_w\oplus\lambda_w}$, where v_w is the wire value on w.
5. **Computing Input Labels.** For each output bit of $\mathcal{F}'^{P'}_{\mathsf{input}}$, if it serves as an input wire w to $\mathtt{Circ}^{S_j}$, $\mathsf{Sim}_{\mathsf{pri}}$ sets $k_{w,v_w\oplus\lambda_w} \| (v_w \oplus \lambda_w)$ as its corresponding input label. Then, $\mathsf{Sim}_{\mathsf{Pri}}$ obtains X.
6. **Generating Garbled Circuits.** For each $j = 1, \ldots, n$:
   (a) For each XOR gate in $\mathtt{Circ}^{S_j}$ with input wires a, b and output wire o, $\mathsf{Sim}_{\mathsf{pri}}$ computes
   $$k_{o,v_o\oplus\lambda_o} \| (v_o \oplus \lambda_o) = (k_{a,v_a\oplus\lambda_a} \| (v_a \oplus \lambda_a)) \oplus (k_{b,v_b\oplus\lambda_b} \| (v_b \oplus \lambda_b)).$$
   (b) For each AND gate g in $\mathtt{Circ}^{S_j}$ with input wire a, b and output wire o, $\mathsf{Sim}_{\mathsf{pri}}$ honestly emulates the random oracle $\mathcal{O}_1$ and computes
   $$\mathsf{ct}^{(g)}_{v_a\oplus\lambda_a, v_b\oplus\lambda_b} = \mathcal{O}_1(k_{a,v_a\oplus\lambda_a} \| (v_a \oplus \lambda_a) \| k_{b,v_b\oplus\lambda_b} \| (v_b \oplus \lambda_b) \| g) \oplus k_{o,v_o\oplus\lambda_o} \| (v_o \oplus \lambda_o).$$
   Then, $\mathsf{Sim}_{\mathsf{pri}}$ samples three random $(\log 40T + 1)$-bit strings as the other three ciphertexts for this gate.
   (c) For each output gate (with index $m = (i-1)\ell^2 + a$ that outputs the a-th bit $s^{S_j}_{i,a}$ of S_j's share of $[\boldsymbol{s}_i]^{(2)}$) of $\mathtt{Circ}^{S_j}$ with input wire w, $\mathsf{Sim}_{\mathsf{pri}}$ computes
   $$\mathsf{ct}_{w,v_w\oplus\lambda_w} = \mathcal{O}_2(k_{w,v_w\oplus\lambda_w} \| (v_w \oplus \lambda_w) \| w) \oplus Y^{S_j}_{m, s^{S_j}_{i,a}}.$$
   Finally, $\mathsf{Sim}_{\mathsf{pri}}$ samples a random $\ell \log 40T$-bit string as $\mathsf{ct}_{w,v_w\oplus\lambda_w\oplus 1}$.
   (d) $\mathsf{Sim}_{\mathsf{pri}}$ obtains (F_j, L_j, d_j) from the generated ciphertexts, wire labels, and output wire masks.
7. **Encrypting Input Labels.** For $i = 1, \ldots, \mathsf{rec}$, the β-th bit $s_{i,\beta}$ of $\boldsymbol{s}_i$ is used as an input wire w in $\mathtt{Circ}^{R_i}$, $\mathsf{Sim}_{\mathsf{pri}}$ honestly emulates the random oracle $\mathcal{O}_1$ and computes
   $$\mathsf{ct}^{(i)}_{w,v_w\oplus\lambda_w} = \mathcal{O}_1(\boldsymbol{r}_{s_{i,\beta},\beta} \| s_{i,\beta} \| i \| \beta \| w) \oplus k_{w,v_w\oplus\lambda_w} \| (v_w \oplus \lambda_w).$$
   Then, $\mathsf{Sim}_{\mathsf{pri}}$ samples a random $(\log 40T + 1)$-bit string as $\mathsf{ct}^{(i)}_{w,v_w\oplus\lambda_w\oplus 1}$. $\mathsf{Sim}_{\mathsf{pri}}$ now gets F_{ct}.

8. **Sampling RO Queries.**
   (a) For each $j = 1, \ldots, n$:
      i. $\mathsf{Sim}_{\mathsf{pri}}$ samples a random $\log 40T$-bit string as $\Delta^{S_j} \neq 0$.
      ii. For each wire w in $\mathtt{Circ}^{S_j}$ that is not an output wire of an output gate. $\mathsf{Sim}_{\mathsf{pri}}$ computes $k_{w,v_w \oplus \lambda_w \oplus 1} = k_{w,v_w \oplus \lambda_w} \oplus \Delta^{S_j}$.
      iii. $\mathsf{Sim}_{\mathsf{pri}}$ initializes a set Q_j containing:
         – $k_{a,\alpha} \| \alpha \| k_{b,\beta} \| \beta \| g$ for each AND gate g in $\mathtt{Circ}^{S_j}$ and each pair of $(\alpha, \beta) \in \{0,1\}^2$ such that $(\alpha, \beta) \neq (v_a \oplus \lambda_a, v_b \oplus \lambda_b)$.
         – $k_{w,v_w \oplus \lambda_w \oplus 1} \| (v_w \oplus \lambda_w \oplus 1) \| w$ for each output gate in $\mathtt{Circ}^{S_j}$ with input wire w.
   (b) For $i = 1, \ldots, \mathsf{rec} + n_o$, $\mathsf{Sim}_{\mathsf{pri}}$ samples k random $\log 40T$-bit strings as $\boldsymbol{r}_{s_{i,\beta} \oplus 1, \beta} = (r^{(1)}_{s_{i,\beta} \oplus 1, \beta}, \ldots, r^{(\log 40T)}_{s_{i,\beta} \oplus 1, \beta})$ for $\beta = 1, \ldots, k$. If $R_i = S_j$, $\mathsf{Sim}_{\mathsf{pri}}$ adds $\boldsymbol{r}_{s_{i,\beta} \oplus 1, \beta} \| (s_{i,\beta} \oplus 1) \| i \| \beta \| w$ into Q_j. If $R_i = P$, $\mathsf{Sim}_{\mathsf{pri}}$ computes the corresponding $\{\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}, \boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)}\}_{\alpha=1}^{\log 40T}$, which are elements in d_{mac}. After this step, $(\tilde{C}, X, d)$ has already been obtained by $\mathsf{Sim}_{\mathsf{pri}}$.
9. **Emulating Randam Oracles.** $\mathsf{Sim}_{\mathsf{pri}}$ invokes the adversary $\mathcal{A}$ with input $(\tilde{C}, X, d)$. In particular, once $\mathcal{A}$ queries a random oracle with query q:
   – If $q \notin Q_j$ for all $j \in \{1, \ldots, n\}$, $\mathsf{Sim}_{\mathsf{pri}}$ honestly emulates the random oracle.
   – If q is in some Q_j, then if $S_j \notin \mathcal{C}$, $\mathsf{Sim}_{\mathsf{pri}}$ sets $\mathsf{Corr}_j = 1$ and adds S_j into $\mathcal{C}$. Then:
      * If $|\mathcal{C}| > \kappa$, $\mathsf{Sim}_{\mathsf{pri}}$ aborts the simulation.
      * Otherwise, $\mathsf{Sim}_{\mathsf{pri}}$ runs Sim_0 to compute S_j's view based on the views other parties in $\mathcal{C}$. Then, for each $i = 1, \ldots, \mathsf{rec}$, $\mathsf{Sim}_{\mathsf{pri}}$ samples S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for the i-th reconstruction based on the shares for servers in $\mathcal{C}$ and computes S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ accordingly by:

$$[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)} = [\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)} \oplus [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}] \otimes [\boldsymbol{s}_i]^{(2)}.$$

     Then the response is determined via the following process:
      * If the queried oracle is $\mathcal{O}_1$, the response to q is determined by the equation for computing a ciphertext, which is of the form

$$\mathsf{ct} = \mathcal{O}_1(q) \oplus \mathsf{m}$$

        for a message m that can be computed from S_j's view and the sampled wire labels (by computing the wire masks). Note that all the ciphertexts have already been sampled. Then, the response can be computed by $\mathsf{ct} \oplus m$. The remaining bits of the output from $\mathcal{O}$ for the same query are then randomly sampled. Then the output of $\mathcal{O}$ for this query is fixed.
      * If the queried oracle is $\mathcal{O}_2$, the response to q is also determined by the equation for computing a ciphertext, which is of the form

$$\mathsf{ct} = \mathcal{O}_2(q) \oplus \mathsf{m}$$

        for a message m that can be computed from S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$. Note that all the ciphertexts have already been sampled. Then, the response can also be computed by $\mathsf{ct} \oplus m$. The remaining bits of the output from $\mathcal{O}$ for the same query are then randomly sampled. Then the output of $\mathcal{O}$ for this query is fixed.

Figure 19: The simulator for proving the privacy.

We construct the following hybrids:

Hyb$_0$: $\mathsf{Sim}_{\mathsf{pri}}$ learns the input x. Then, $\mathsf{Sim}_{\mathsf{pri}}$ runs Garble on inputs $(1^\kappa, C)$ to generate $(\tilde{C}, e, d)$ and computes X by $\mathsf{Encode}(e, x)$. Finally, $\mathsf{Sim}_{\mathsf{pri}}$ outputs $(\tilde{C}, X, d)$. While interacting with the adversary $\mathcal{A}$, $\mathsf{Sim}_{\mathsf{pri}}$ emulates the random oracles honestly. This corresponds to the real-world scenario.

Hyb$_1$: In this hybrid, for each $j = 1, \ldots, n$, $\mathsf{Sim}_{\mathsf{pri}}$ does not follow the algorithm description to sample each λ_w for each wire w in $\mathtt{Circ}^{S_j}$. Instead, he samples $v_w \oplus \lambda_w$ randomly and computes $\lambda_w = (v_w \oplus \lambda_w) \oplus v_w$, where v_w is obtained by following the execution of Π_0. Since λ_w is a random bit, so is $v_w \oplus \lambda_w$. We only change the order of generating λ_w and $v_w \oplus \lambda_w$ without changing their distributions. Thus, **Hyb**$_1$ and **Hyb**$_0$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, for each $j = 1, \ldots, n$, $\mathsf{Sim}_{\mathsf{pri}}$ does not follow the algorithm description to sample $k_{w,0}$ for each wire w with $v_w \oplus \lambda_w = 1$ in $\mathtt{Circ}^{S_j}$ excluding output wires for XOR/output gates. Instead, $\mathsf{Sim}_{\mathsf{pri}}$ samples $k_{w,1}$ for these wires first and then computes $k_{w,0} = k_{w,1} \oplus \Delta^{S_j}$. Since $k_{w,0}$ is a random $\log 40T$-bit string, so is $k_{w,1}$. We only change the order of sampling $k_{w,0}$ and $k_{w,1}$ for these wires without changing their distributions. Thus, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ have the same output distribution.

$\mathbf{Hyb}_3$: In this hybrid, for each $j = 1, \ldots, n$ and each AND gate (where we use the same notation as in the description of Garble) of $\mathtt{Circ}^{S_j}$, $\mathsf{Sim}_{\mathsf{pri}}$ doesn't follow the algorithm description to emulate the random oracle and compute the ciphertexts corresponding to $(\alpha, \beta) \neq (v_a \oplus \lambda_a, v_b \oplus \lambda_b)$. Instead, $\mathsf{Sim}_{\mathsf{pri}}$ directly samples $(\log 40T + 1)$-bit strings as these ciphertexts. Then $\mathsf{Sim}_{\mathsf{pri}}$ computes the output of $\mathcal{O}_1$ by

$$\mathcal{O}_1(k_{a,\alpha} \| \alpha \| k_{b,\beta} \| \beta \| g) = \mathsf{ct}^{(g)}_{\alpha,\beta} \oplus k_{o,\chi_g(\alpha,\beta)} \| \chi_g(\alpha, \beta).$$

The remaining bits of the output from $\mathcal{O}$ for the same query are then randomly sampled. Then the output of $\mathcal{O}$ for this query is fixed. Note that all the random oracle queries are different in Garble. The outputs of $\mathcal{O}$ to these queries are uniformly random. Therefore, the ciphertexts are also uniformly random. We only change the order of generating $\mathcal{O}_1(k_{a,\alpha} \| \alpha \| k_{b,\beta} \| \beta \| g)$ and $\mathsf{ct}^{(g)}_{\alpha,\beta}$ for these gates and wire values without changing their distributions. Thus, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have the same output distribution.

$\mathbf{Hyb}_4$: In this hybrid, for each $j = 1, \ldots, n$ and each output gate (where we use the same notation as in the description of Garble) of $\mathtt{Circ}^{S_j}$, $\mathsf{Sim}_{\mathsf{pri}}$ doesn't follow the algorithm description to emulate the random oracle and compute the ciphertexts corresponding to output wire value $v_w \oplus \lambda_w \oplus 1$. Instead, $\mathsf{Sim}_{\mathsf{pri}}$ directly samples $\ell \log 40T$-bit strings as the ciphertext. Then $\mathsf{Sim}_{\mathsf{pri}}$ computes the output of $\mathcal{O}_2$ by

$$\mathcal{O}_2(k_{w,v_w \oplus \lambda_w \oplus 1} \| (v_w \oplus \lambda_w \oplus 1) \| w) = \mathsf{ct}_{w,v_w \oplus \lambda_w \oplus 1} \oplus Y^{S_j}_{m, s^{S_j}_{i,a} \oplus 1}.$$

The remaining bits of the output from $\mathcal{O}$ for the same query are then randomly sampled. Then the output of $\mathcal{O}$ for this query is fixed. Since the output of $\mathcal{O}$ to the query is uniformly random, so is the ciphertext. We only change the order of generating $\mathcal{O}_2(k_{w,v_w \oplus \lambda_w \oplus 1} \| (v_w \oplus \lambda_w \oplus 1) \| w)$ and $\mathsf{ct}_{w,v_w \oplus \lambda_w \oplus 1}$ for the output gate without changing their distributions. Thus, $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ have the same output distribution.

$\mathbf{Hyb}_5$: In this hybrid, for $i = 1, \ldots, \mathsf{rec}$, the β-th bit $s_{i,\beta}$ of $\boldsymbol{s}_i$ is used as an input wire w in $\mathtt{Circ}^{R_i}$, $\mathsf{Sim}_{\mathsf{pri}}$ does not follow the algorithm description to emulate the random oracle and compute $\mathsf{ct}^{(i)}_{w,v_w \oplus \lambda_w \oplus 1}$. Instead, $\mathsf{Sim}_{\mathsf{pri}}$ samples a random $(\log 40T + 1)$-bit string as $\mathsf{ct}^{(i)}_{w,v_w \oplus \lambda_w \oplus 1}$ and computes

$$\mathsf{ct}^{(i)}_{w,v_w \oplus \lambda_w \oplus 1} = \mathcal{O}_1(\boldsymbol{r}_{s_{i,\beta} \oplus 1, \beta} \| (s_{i,\beta} \oplus 1) \| i \| \beta \| w) \oplus k_{w,v_w \oplus \lambda_w \oplus 1} \| (v_w \oplus \lambda_w \oplus 1).$$

The remaining bits of the output from $\mathcal{O}$ for the same query are then randomly sampled. Then the output of $\mathcal{O}$ for this query is fixed. Since the output of $\mathcal{O}$ to the query is uniformly random, so is this ciphertext. We only change the order of generating $\mathsf{ct}^{(i)}_{w,v_w \oplus \lambda_w \oplus 1}$ and $\mathcal{O}_1(\boldsymbol{r}_{s_{i,\beta} \oplus 1, \beta} \| (s_{i,\beta} \oplus 1) \| i \| \beta \| w)$ without changing their distributions. Thus, $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have the same output distribution.

$\mathbf{Hyb}_6$: In this hybrid, for each $j = 1, \ldots, n$ $\mathsf{Sim}_{\mathsf{pri}}$ additionally initializes a set Q_j containing:

- $k_{a,\alpha} \| \alpha \| k_{b,\beta} \| \beta \| g$ for each AND gate g in $\mathtt{Circ}^{S_j}$ and each pair of $(\alpha, \beta) \in \{0,1\}^2$ such that $(\alpha, \beta) \neq (v_a \oplus \lambda_a, v_b \oplus \lambda_b)$.
- $k_{w,v_w \oplus \lambda_w \oplus 1} \| (v_w \oplus \lambda_w \oplus 1) \| w$ for each output gate in $\mathtt{Circ}^{S_j}$ with input wire w.

For $i = 1, \ldots, \mathsf{rec}$, $\mathsf{Sim}_{\mathsf{pri}}$ adds $\boldsymbol{r}_{s_{i,\beta} \oplus 1, \beta} \| (s_{i,\beta} \oplus 1) \| i \| \beta \| w$ into Q_j. $\mathsf{Sim}_{\mathsf{pri}}$ also initializes $\mathcal{C} = \emptyset$ and $\mathsf{corr}_j = 0$ for each $j = 1, \ldots, n$. This does not affect the output distribution. Thus, $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ have the same output distribution.

$\mathbf{Hyb}_7$: In this hybrid, for each $i = 1, \ldots, \mathsf{rec}$ and $\beta = 1, \ldots, k$, $\mathsf{Sim}_{\mathsf{pri}}$ does not follow the algorithm description to sample the corresponding strings $\boldsymbol{r}_{0,\beta}, \boldsymbol{r}_{1,\beta}$. Instead, $\mathsf{Sim}_{\mathsf{pri}}$ first samples $\boldsymbol{r}_{s_{i,\beta},\beta}$ randomly and then samples $\boldsymbol{r}_{s_{i,\beta} \oplus 1, \beta}$ randomly. $\boldsymbol{r}_{0,\beta}, \boldsymbol{r}_{1,\beta}$ are then computed from $\boldsymbol{r}_{s_{i,\beta},\beta}, \boldsymbol{r}_{s_{i,\beta} \oplus 1, \beta}$. Since each bit of $\boldsymbol{r}_{0,\beta}, \boldsymbol{r}_{1,\beta}$ is still sampled randomly, we do not change the output distribution. Thus, $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have the same output distribution.

$\mathbf{Hyb}_8$: In this hybrid, $\mathsf{Sim}_{\mathsf{pri}}$ does not follow the algorithm description to generate $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ for each reconstruction of $\Sigma^{(2)}$-sharing. Instead, $\mathsf{Sim}_{\mathsf{pri}}$ first generate $\{[\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ and then compute $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ by:

$$[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)} = [\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)} \oplus [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}] \otimes [\boldsymbol{s}_i]^{(2)}.$$

Note that $[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}$ is randomly sampled based on the secret, $[\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)}$ is also random based on the secret. We only change the order of generating $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ and $[\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)}$ for each reconstruction of $\Sigma^{(2)}$-sharing without changing their distributions. Thus, $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ have the same output distribution.

$\mathbf{Hyb}_9$: In this hybrid, while emulating $\mathcal{O}$ to interact with $\mathcal{A}$, once a query $q \in Q_j$ is received, $\mathsf{Sim}_{\mathsf{pri}}$ adds S_j into $\mathcal{C}$ if $\mathsf{corr}_j = 0$ and sets $\mathsf{corr}_j = 1$. Whenever $|\mathcal{C}| > \kappa$ before $\mathcal{A}$ terminates, $\mathsf{Sim}_{\mathsf{pri}}$ aborts the simulation. Note that changing $\mathcal{C}$ doesn't affect the output distribution. The only case that will change the output distribution happens when $|\mathcal{C}| > \kappa$, i.e., $\mathcal{A}$ queries at least one element in Q_j for more than κ $j \in \{1, \ldots, n\}$. Note that the inputs to $\mathcal{A}$ are not computed with Δ^{S_j} for each $j = 1, \ldots, n$ and $\boldsymbol{r}_{s_{i,\beta} \oplus 1, \beta}$ for each $i = 1, \ldots, \mathsf{rec}$ and $\beta = k$, the queries made by $\mathcal{A}$ are independent of them. Note that the elements in Q_j either contain a string added by a randomly sampled $\log 40T$-bit string Δ^{S_j} or contain a randomly sampled $\log 40T$-bit string $\boldsymbol{r}_{s_{i,\beta} \oplus 1, \beta}$. That is to say, each query is either out of Q_j or it can be used to compute a random $\log 40T$-bit string correctly by a deterministic process. Thus, T queries to the random oracle can at most compute T guesses for the random string by the process, while the string has $2^{\log 40T} = 40T$ possible values. Thus, the adversary can query the elements in Q_j with no more than $1/40$ probability.

Let X_j be the random variable corresponding to whether the RO queries made by $\mathcal{A}$ contain an element in Q_j. If true, X_j is set to be 1. Otherwise, it is set to be 0. The random variable is distributed with respect to the randomly sampled Δ^{S_j} and each $\boldsymbol{r}_{s_{i,\beta} \oplus 1, \beta}$ with $R_i = S_j$, and the sampling processes of these random strings for $j = 1, \ldots, n$ are independent. Thus, $X_1, \ldots, X_n$ are n independent random variables taking values in $\{0, 1\}$, where each $\mathbb{E}(X_j) \leq 1/40$. Thus, for $X = X_1 + \cdots + X_n$, we have $\mathbb{E}(X) \leq n/T$. By the Chernoff bound, we have:

$$P(X > \kappa) \leq P\left(X \geq (1 + \frac{40-4}{4}) \cdot \frac{n}{40}\right) \leq e^{-\frac{(40-4)^2 \kappa}{40(40+4)}} < 2^{-\kappa}.$$

Therefore, the distribution changes with only $< 2^{-\kappa}$ probability. Thus, the distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ are indistinguishable for $\mathcal{A}$.

$\mathbf{Hyb}_{10}$: In this hybrid, while emulating $\mathcal{O}$ to interact with $\mathcal{A}$, $\mathsf{Sim}_{\mathsf{pri}}$, for each j such that no element in Q_j is queried by $\mathcal{A}$, $\mathsf{Sim}_{\mathsf{pri}}$ does not use the ciphertexts and the view of S_j in Π_0 to compute the oracle outputs corresponding to queries in Q_j. Instead, $\mathsf{Sim}_{\mathsf{pri}}$ honestly emulates $\mathcal{O}$ while answering these queries. Since these queries are never called by $\mathcal{A}$, we do not change the output distribution. Thus, $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ have the same output distribution.

Note that we do not need the view of each S_j out of $\mathcal{C}$ during the simulation (excluding their shares for the reconstructed $\Sigma^{(2)}$-sharings), $\mathsf{Sim}_{\mathsf{pri}}$ no longer generates them while following Π_0 to compute the wire values in future hybrids.

$\mathbf{Hyb}_{11}$: In this hybrid, $\mathsf{Sim}_{\mathsf{pri}}$ does not follow the execution of Π_0 to generate the views of P and the servers in $\mathcal{C}$. Instead, $\mathsf{Sim}_{\mathsf{pri}}$ runs Sim_0 to generate them. While emulating $\mathcal{F}$ while running Sim_0, $\mathsf{Sim}_{\mathsf{pri}}$ sends $\mathsf{En}(C(x) \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}))$ to Sim_0 as $\mathcal{F}$'s output. In particular, the view of P and all the reconstructed $\Sigma^{(2)}$-sharings are generated at the beginning. Then, whenever $\mathcal{A}$ queries an element in Q_j for the first time, $\mathsf{Sim}_{\mathsf{pri}}$ runs Sim_0 to generate S_j's view based on the views of P and the servers that have been included in $\mathcal{C}$ before S_j. Since $|\mathcal{C}|$ is guaranteed to be no more than $t = \kappa$, by the property of Sim_0, the generated views generated by Sim_0 distribute the same as the computed views from the execution of Π_0. Thus, $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ have the same output distribution.

$\mathbf{Hyb}_{12}$: In this hybrid, $\mathsf{Sim}_{\mathsf{pri}}$ does not follow the algorithm description to generate $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for each reconstruction of $\Sigma^{(2)}$-sharing. Instead, whenever $\mathcal{A}$ queries an element in Q_j for the first time, $\mathsf{Sim}_{\mathsf{pri}}$ samples S_j's share of these sharings based on the shares for the servers that have been included in $\mathcal{C}$ before S_j. The shares for servers out of $\mathcal{C}$ are not used in the simulation, so $\mathsf{Sim}_{\mathsf{pri}}$ no longer samples

them. We can regard these shares as sampled based on the shares for servers in $\mathcal{C}$ and the secret. Finally, the shares for servers out of $\mathcal{C}$ are removed. Since $|\mathcal{C}|$ is guaranteed to be no more than $t = \kappa$, S_j's share can be generated independently of the secret, so we only change the order of generating the servers' shares without changing their distributions. Since the shares for servers out of $\mathcal{C}$ are not used in the simulation, not generating them does not affect the output distribution either. Thus, $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ have the same output distribution.

Note that $\mathbf{Hyb}_{12}$ is the ideal-world scenario, and the total variation distance of the distributions of $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_{12}$ is smaller than $2^{-\kappa}$, so our garbling scheme satisfies the privacy.

F.2 Proof of Obliviousness

The construction of $\mathsf{Sim}_{\mathsf{obl}}$ is quite similar as $\mathsf{Sim}_{\mathsf{pri}}$. It only deviates from the construction of $\mathsf{Sim}_{\mathsf{pri}}$ in the following points:

1. $\mathsf{Sim}_{\mathsf{obl}}$ does not take $C(x)$ as a part of its input.
2. While simulating the initialization of Π_0, P is set to be honest, and Step 2 of $\mathsf{Sim}_{\mathsf{pri}}$ is no longer performed.
3. At the end of Step 8 of $\mathsf{Sim}_{\mathsf{pri}}$, $\mathsf{Sim}_{\mathsf{obl}}$ no longer computes d_{mac}.
4. In Step 9, $\mathsf{Sim}_{\mathsf{obl}}$ only inputs $(\tilde{C}, X)$ to $\mathcal{A}$.

The other steps of $\mathsf{Sim}_{\mathsf{obl}}$ are all the same as $\mathsf{Sim}_{\mathsf{pri}}$. As these changed steps do not influence the other steps, the output distribution of $(\tilde{C}, X)$ and the emulation strategy of the random oracles remain unchanged. Thus, the proof of privacy implies the obliviousness.

F.3 Proof of Authenticity

If the garbling scheme does not have authenticity, i.e., the distribution outputs `TRUE` with more than $2^{-\kappa}$ probability in the real world, then by obliviousness, the distribution must output `TRUE` with more than $2^{-\kappa}$ probability when $\mathcal{A}$ is interacting with $\mathsf{Sim}_{\mathsf{obl}}$. However, during the whole execution of $\mathsf{Sim}_{\mathsf{obl}}$, $\mathsf{Sim}_{\mathsf{obl}}$ does not need to generate $\{\boldsymbol{r}^{(\alpha)}_{\boldsymbol{s}_i \oplus \mathbf{1}}\}_{\alpha=1}^{\log 40T}$ for $i = \mathsf{rec}+1, \ldots, \mathsf{rec}+n_o$. Note that in Garble, each bit of $\{\boldsymbol{r}^{(\alpha)}_{\boldsymbol{s}_i \oplus \mathbf{1}}\}_{\alpha=1}^{\log 40T}$ are uniformly randomly sampled for these reconstructions. If $\mathcal{A}$ interact with $\mathsf{Sim}_{\mathsf{obl}}$, $\mathcal{A}$ only has no more than $2^{-40\sigma \log T} < T/2^{\sigma}$ probability to output $Y = (\kappa, \{\boldsymbol{s}'_i, \boldsymbol{r}^{(1)}_{\boldsymbol{s}'_i} \ldots, \boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}'_i}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o})$ such that the check in Step 3 of Decode passes, but $\{\boldsymbol{s}'_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ flips σ bits from $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$. Here $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ are obtained by honestly computing Eval.

Note that the output of $C(x)$ is encoded using Σ-sharings, where Σ is an LSSS with 3-multiplicative reconstruction, which is strong multiplicative, i.e., the secret of a $\Sigma^{(2)}$-sharing is uniquely determined by any $n - t$ shares. This follows from Lemma 4 given below, which is proved in [ZLC+08].

Lemma 4 ([ZLC+08]). *If an (n, t, k, ℓ)-LSSS Σ has 3-multiplicative reconstruction with $\Sigma^{(2)}, \Sigma^{(3)}$, then the secret of a $\Sigma^{(2)}$-sharing is uniquely determined by any $n - t$ shares.*

Therefore, the adversary should at least flip $t + 1$ bits of the output from $\overline{C}$, i.e., $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$, which only success with probability $< 2^{-t \log T} = 1/2^t T < 2^{-\kappa}$, which leads to a contradiction. Thus, our garbling scheme satisfies the authenticity.

G Cost Analysis for the Garbling Scheme

We analyze the garbled circuit size and the input label size of our garbling scheme.

Garbled Circuit Size. Recall that the garbled circuit $\tilde{C}$ contains:

- The lists $F_1, \ldots, F_n$ of ciphertexts, where each F_j contains $O(1)$ ciphertexts of size $O(\log T)$ for each AND/output gate of $\mathtt{Circ}^{S_j}$.
- The lists $L_1, \ldots, L_n$ of labels for preprocessing data and input data for P'', where an $O(\log T)$-bit label is contained in them for each bit of output from $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^{P''}$.
- The lists $d_1, \ldots, d_n$ of output wire masks for $\mathtt{Circ}^{S_1}, \ldots, \mathtt{Circ}^{S_n}$.

From Lemma 2 we know that the communication cost of Π_0 is $O(|\overline{C}|+Dn^2)$ bits, the total number of AND gates in $\mathtt{Circ}^{S_1}, \ldots, \mathtt{Circ}^{S_n}$ is $O(|\overline{C}|+Dn^2)$, and the total output size of $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}$ is $O(|\overline{C}|+W_O+Dn^2) = O(|\overline{C}|+Dn^2)$. Recall that $\overline{C}$ only adds linear computations over C, and the circuit size is only a constant fraction larger than C, i.e., $|\overline{C}| = O(|C|)$. The number of parties $n = O(\kappa)$, and the total number of output gates and output wires of $\mathtt{Circ}^{S_1}, \ldots, \mathtt{Circ}^{S_n}$ is exactly the communication cost of Π_0. Thus:

- The total size of $F_1, \ldots, F_n$ is $O(\log T)$ times the total number of AND/output gates in $\mathtt{Circ}^{S_1}, \ldots, \mathtt{Circ}^{S_n}$, which is $O(|C| \log T + D\kappa^2 \log T)$ bits.
- The total size of $L_1, \ldots, L_n$ is $O(\log T)$ times the output size of $\mathcal{F}_{\mathsf{prep}}$, which is also $O(|C| \log T + D\kappa^2 \log T)$ bits.
- The total size of $d_1, \ldots, d_n$ is the total number of output wires in $\mathtt{Circ}^{S_1}, \ldots, \mathtt{Circ}^{S_n}$, which is $O(|C| + Dn^2)$ bits.

Thus, the garbled circuit size is $|\tilde{C}| = O(|C| \log T + D\kappa^2 \log T)$ bits.

Input Label Size. Recall that the output of Encode is the wire labels for the output of $\mathcal{F}_{\mathsf{input}}^{P'}$. From Lemma 2, the output size of $\mathcal{F}_{\mathsf{input}}^{P'}$ is $O(W_I)$, and each bit of the output has a wire label of size $O(\log T)$. Therefore, the total input label size is $O(W_I \log T)$ bits.

H Detailed Evaluation Process in Our 2PC

We give the detailed evaluation process in our maliciously secure 2PC here.

The Evaluation Process. E separates $\mathtt{Circ}^{S_j}$ to $\mathtt{Circ}_1^{S_j}, \ldots, \mathtt{Circ}_{\mathsf{rec}}^{S_j}$, where rec is the number of reconstructions in the online phase of Π_0'. Each $\mathtt{Circ}_i^{S_j}$ outputs S_j's share of the i-th $\Sigma^{(2)}$-sharing reconstructed in Π_0'. E then evaluates the garbled circuit as follows:

1. For each $j = 1, \ldots, n$, all the input labels for $\mathtt{Circ}_1^{S_j}$ have already been prepared. Then, E evaluates $\mathtt{Circ}_1^{S_j}$ gate by gate following Step 2 of Eval.
2. E computes $[\boldsymbol{r}_{\boldsymbol{s}_1}^{(\alpha)}]^{(3)}$ for $\alpha = 1, \ldots, 2\log T$ from the output labels computed in the last step. Then, E reconstructs $\{\boldsymbol{r}_{\boldsymbol{s}_1}^{(\alpha)}\}_{\alpha=1}^{2\log T}$ using the error correction algorithm.
3. For each $j = 1, \ldots, n$ and input wire w of output gates in $\mathtt{Circ}_1^{S_j}$, parse λ_w from $\tilde{C}$. Then, compute $v_w = (v_w \oplus \lambda_w) \oplus \lambda_w$. Compute the sharing $[\boldsymbol{s}_1]^{(2)}$ from all these output wire values. Then, E reconstructs $\boldsymbol{s}_1$ using the error correction algorithm.
4. If the η-th bit $s_{1,\eta}$ of $\boldsymbol{s}_1$ is used as an input wire w in $\mathtt{Circ}^{R_1}$, E computes
$$k_{w,s_{1,\eta}\oplus\lambda_w} \| (s_{1,\eta} \oplus \lambda_w) = \mathsf{ct}_{w,s_{1,\eta}}^{(1)} \oplus \mathcal{O}_1(\boldsymbol{r}_{s_{1,\eta},\eta} \| s_{1,\eta} \| 1 \| \eta \| w).$$
5. E evaluates $\mathtt{Circ}_2^{S_j}, \ldots, \mathtt{Circ}_{\mathsf{rec}}^{S_j}$ successively in the same way as in Steps (a)-(d) for all $j = 1, \ldots, n$. Eventually, $\boldsymbol{s}_i$ and their corresponding $\{[\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{2\log T}$ are obtained for all $i = \mathsf{rec}+1, \ldots, \mathsf{rec}+n_o$ where $R_i = G$. E also obtains the output sharings for his own output wires.

I Security Proof for the 2PC

We prove the security of Π by constructing an ideal adversary Sim. Then we will show that the output in the ideal world is indistinguishable from that in the real world against the computationally unbounded adversary $\mathcal{A}$ who has T RO queries using hybrid arguments. Note that $\mathcal{A}$ may corrupt either the garbler G or the evaluator E, so we prove the security under the two cases separately.

I.1 Security against a Corrupted Garbler

When G is corrupted, we construct the ideal adversary Sim below. We separate the description of Sim into the simulators $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3, \mathsf{Sim}_4$ for the 4 rounds, respectively.

Simulator Sim_1

1. Sim initializes $\mathsf{ShCheck} = \mathsf{CompCheck} = 0$ and $\mathsf{Comp}_1 = \cdots = \mathsf{Comp}_n = 0$.
2. Sim follows the protocol to initialize the protocol Π_0'.
3. For each input wire of index i attached to E, Sim emulates $\mathcal{F}_{\mathsf{OT}}$ to receive $\{X_{i,0}^{S_j}\}_{j=1}^n, \{X_{i,1}^{S_j}\}_{j=1}^n$ from G.
4. Sim emulates $\mathcal{F}_{\mathsf{OT}}$ to receive the challenge s from G.
5. For each bit b (indexed i) of $[\boldsymbol{r}_0]_{\times\kappa'}$, Sim emulates $\mathcal{F}_{\mathsf{OT}}$ to receive $L_{i,0}, L_{i,1} \in \mathbb{F}_2^{\log 40T}$ from G.
6. Sim emulates $\mathcal{O}$ to receive $\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}$ for each $j = 1, \ldots, n$ and $\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}$ for each pair of servers (S_i, S_j). Sim honestly emulates $\mathcal{O}$ to send the outputs to G. Whenever the first $\kappa + 2\log T$ bits of the sampled outputs of $\mathcal{O}$ for two different queries are the same, Sim aborts the simulation.
7. Sim emulates $\mathcal{F}_{\mathsf{OT}}$ to receive $c_1, \ldots, c_n$ and $\sigma_{i,j}$ for each pair of (S_i, S_j) from G. Then, Sim retrieve the corresponding committed $(\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j})$ for each $j = 1, \ldots, n$ and $(\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)})$ for each pair of servers (S_i, S_j) from the RO queries that match the commitments. If such committed inputs and randomness are not found from the RO queries for some commitment, Sim sets the committed values corresponding to this commitment to $\perp$.

Figure 20: The simulator for the first round of Π when G is corrupted.

Simulator Sim_2

1. Sim samples a random $\Sigma_{\times\kappa'}$-sharing as $[\boldsymbol{v}]_{\times\kappa'}$ and honestly emulates $\mathcal{O}$ to compute $\mathcal{O}([\boldsymbol{v}]_{\times\kappa'})$ and obtain $\mathcal{U}$.
2. Sim randomly samples the shares for the servers in $\mathcal{U}$ of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ and computes their shares of $[\boldsymbol{r}_0]_{\times\kappa'}$ by
$$[\boldsymbol{r}_0]_{\times\kappa'} = [\boldsymbol{v}]_{\times\kappa'} - \sum_{i=1}^{k_0} \theta_i [\boldsymbol{I}_i]_{\times\kappa'}.$$
Then Sim computes the label for each bit of these shares from the labels received by $\mathcal{F}_{\mathsf{OT}}$ and follows the protocol to compute the output labels of $\mathtt{RanLin}_j$ for each $S_j \in \mathcal{U}$.
3. Sim samples a random $(\kappa + \log T)$-bit string as s'.
4. Sim sends $(\mathcal{U}, [\boldsymbol{v}]_{\times\kappa'}, s')$ and the output labels for of $\mathtt{RanLin}_j$ for each $S_j \in \mathcal{U}$ to G on behalf of E.

Figure 21: The simulator for the second round of Π when G is corrupted.

Simulator Sim_3

1. For each $j = 1, \ldots, n$, Sim follows the protocol to compute (F_j, L_j, d_j) and the ciphertexts in F_{ct} generated in Step 6 of Garble for $R_i = S_j$ from $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$ received by $\mathcal{O}$.
2. Sim honestly emulates $\mathcal{O}$ to receive $\mathsf{GC}\|\mathsf{VerSh}$ from G and outputs $\mathcal{V}$ to G.
3. Sim receives $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}\}_{j\in\mathcal{V}}, \{\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}, \mathsf{Sh}_{j,i}, \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}\}_{i\in\mathcal{V}, j\in\{1,\ldots,n\}\setminus\{i\}})$ from G on

behalf of E. Then for each $j = 1, \ldots, n$, Sim checks whether:

- (F_j, L_j, d_j) and the ciphertexts in F_{ct} for $R_i = S_j$ match what he computed in the last step.
- The input labels received by $\mathcal{F}_{\mathsf{OT}}$ match View^{S_j}.
- $\mathsf{View}^{S_j}, \boldsymbol{r}^{S_j}_{\mathsf{commit}}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$ received in this step match what he received in the first round while emulating $\mathcal{O}$ if $j \in \mathcal{V}$.
- For the output $\Sigma^{(2)}$-sharings from $\mathcal{F}_{\mathsf{prep}}$ to S_j where S_j holds the secrets, the shares from $\{\mathsf{Sh}_{j,i}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$ form valid sharings, and the secret in View^{S_j} matches the sharings.
- S_j's shares of the four sharings in VerSh are computed honestly from the shares in $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$.

If not, Sim sets $\mathsf{Comp}_j = 1$.

4. If for more than $t/2$ indices $j \in \{1, \ldots, n\}$, it satisfies $\mathsf{Comp}_j = 1$, then Sim sets $\mathsf{CompCheck} = 1$.
5. If $\mathsf{CompCheck} = 0$, for each j with $\mathsf{Comp}_j = 0$, Sim obtains S_j's shares of $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, and $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$ from what G sends to $\mathcal{O}$ in the first round. Sim then checks whether these servers' shares of these sharings are all from valid $\Sigma/\Sigma^{(2)}/\Sigma^{(3)}$-sharings and $\Sigma^{(2)}$-sharings with all-zero secrets. If not, Sim sets $\mathsf{ShCheck} = 1$.

Figure 22: The simulator for the third round of Π when G is corrupted.

Simulator Sim_4

1. Sim follows the protocol to verify the garbled circuit, excluding the check on input labels from $\mathcal{F}_{\mathsf{OT}}$. If the check does not pass, Sim aborts the protocol on behalf of E.
2. For the input labels from $\mathcal{F}_{\mathsf{OT}}$:
   - If for more than κ invocations $\mathcal{F}_{\mathsf{OT}}$, the labels of P_j's input shares received by $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{O}$ do not match among all $j \in \mathcal{V}$, Sim aborts the protocol on behalf of E.
   - If only for $\leq \kappa$ invocations $\mathcal{F}_{\mathsf{OT}}$, the labels of P_j's input shares received by $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{O}$ do not match among all $j \in \mathcal{V}$ for these input wires, Sim randomly generates the $\leq \kappa$ shares containing these OT input bits based on the shares for servers in $\mathcal{U}$ and follows the protocol to check whether the input labels (corresponding to the sampled bits) received by $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{O}$ match. If not, Sim aborts the protocol on behalf of E.
3. If $\mathsf{ShCheck} = 1$ or $\mathsf{CompCheck} = 1$, Sim aborts the simulation.
4. Sim reconstructs $\mathsf{mask}, \boldsymbol{r}^{(1)}_{\mathsf{en}}, \ldots, \boldsymbol{r}^{(W_O/k)}_{\mathsf{en}}$ from the outputs of $\mathcal{F}^G_{\mathsf{input}}$ (received by $\mathcal{O}$) using error correction. Then, Sim uses then and the outputs of C for G to compute G's outputs of $\overline{C}$, i.e. $\{\boldsymbol{s}_i\}^{\mathsf{rec}+n_o}_{i=\mathsf{rec}+1}$. Sim then computes the corresponding $\boldsymbol{r}^{(\alpha)}_{\boldsymbol{s}_i} = \boldsymbol{r}^{(\alpha)}_{\boldsymbol{0}} + (\boldsymbol{r}^{(\alpha)}_{\boldsymbol{1}} - \boldsymbol{r}^{(\alpha)}_{\boldsymbol{0}}) * \boldsymbol{s}_i$ for each $\alpha = 1, \ldots, \log 40T$, where $\boldsymbol{r}^{(\alpha)}_{\boldsymbol{0}}, \boldsymbol{r}^{(\alpha)}_{\boldsymbol{1}}$ are reconstructed from the sharings $[\boldsymbol{r}^{(\alpha)}_{\boldsymbol{0}}]^{(3)}, [\boldsymbol{r}^{(\alpha)}_{\boldsymbol{1}} - \boldsymbol{r}^{(\alpha)}_{\boldsymbol{0}}]$ received by $\mathcal{O}$. Then, Sim sends $\{\boldsymbol{s}_i, \boldsymbol{r}^{(1)}_{\boldsymbol{s}_i}, \ldots, \boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}_i}\}^{\mathsf{rec}+n_o}_{i=\mathsf{rec}+1}$ to G on behalf of E.
5. Sim outputs what $\mathcal{A}$ outputs.

Figure 23: The simulator for the fourth round of Π when G is corrupted.

We construct the following hybrids:

$\mathbf{Hyb}_0$: In this hybrid, Sim gets E's inputs and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, Sim additionally sets $\mathsf{ShCheck} = \mathsf{CompCheck} = 0$ and $\mathsf{Comp}_1 = \cdots = \mathsf{Comp}_n = 0$ at the beginning of the simulation. This does not affect the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, while emulating the random oracle $\mathcal{O}$, Sim aborts the simulation if the first $\kappa + 2 \log T$ bits of the RO outputs are the same for two different queries. Then, for each commitment generated by $\mathcal{O}$ from G, Sim can extract the committed inputs and randomness from the queries made by G. If such a query that matches the commitment is not found, Sim sets the committed inputs and

randomness to $\perp$. Note that the RO output for each query is sampled randomly, so the probability that the first $\kappa + 2\log T$ bits of two outputs are the same is $1/2^{-(\kappa+2\log T)}$. Taking the union bound of all $< T^2$ possible pairs of queries, the probability is still less than $2^{-}\kappa$. Thus, the distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are indistinguishable for $\mathcal{A}$.

Note that the sharing $[\boldsymbol{r}_0]_{\times\kappa'}$ is not used for generating any message in Round 1, Sim delays the generation of them into Round 2.

$\mathbf{Hyb}_3$: In this hybrid, Sim does not follow the protocol to sample $[\boldsymbol{r}_0]_{\times\kappa'}$ and compute $[\boldsymbol{v}]_{\times\kappa'}$. Instead, Sim samples $[\boldsymbol{v}]_{\times\kappa'}$ first and then computes $[\boldsymbol{r}_0]_{\times\kappa'}$ by

$$[\boldsymbol{r}_0]_{\times\kappa'} = [\boldsymbol{v}]_{\times\kappa'} - \sum_{i=1}^{k_0} \theta_i [\boldsymbol{I}_i]_{\times\kappa'}.$$

Since $[\boldsymbol{r}_0]_{\times\kappa'}$ is a uniformly random $\Sigma_{\times\kappa'}$-sharing, so is $[\boldsymbol{v}]_{\times\kappa'}$. Therefore, we only change the order of them without changing their distribution. Thus, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have the same output distribution.

Note that the input sharings $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ are not used for generating any message in Round 1, Sim delays the generation of them into Round 2 after the sharing $[\boldsymbol{v}]_{\times\kappa'}$ is generated.

$\mathbf{Hyb}_4$: In this hybrid, while generating $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$, Sim first generates the shares for servers in $\mathcal{U}$ and then samples the other servers' shares based on these shares and the secrets. Note that $|\mathcal{U}| < t$, the shares for servers in $\mathcal{U}$ are independent of the secrets. Therefore, we only change the order of the generations for different servers' shares without changing their distributions. Thus, $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ have the same output distribution.

$\mathbf{Hyb}_5$: In this hybrid, after receiving $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}\}_{j\in\mathcal{V}})$ from G on behalf of E, for each $j = 1, \ldots, n$, Sim additionally checks whether:

- (F_j, L_j, d_j) and the ciphertexts in F_{ct} for $R_i = S_j$ match View^{S_j} and the sets $\{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$ received by $\mathcal{O}$ when they are committed.
- The input labels received by $\mathcal{F}_{\mathsf{OT}}$ match View^{S_j}.
- $\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}$ received in this step match what he received while emulating $\mathcal{O}$.
- For the output $\Sigma^{(2)}$-sharings from $\mathcal{F}_{\mathsf{prep}}$ to S_j where S_j holds the secrets, the shares from $\{\mathsf{Sh}_{j,i}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$ form valid sharings, and the secret in View^{S_j} matches the sharings.
- S_j's shares of the four sharings in VerSh are computed honestly from the shares in $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$.

If not, Sim sets $\mathsf{Comp}_j = 1$. If for more than $t/2$ indices $j \in \{1, \ldots, n\}$, it satisfies $\mathsf{Comp}_j = 1$, then Sim sets $\mathsf{CompCheck} = 1$. This does not affect the output distribution. Thus, $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have the same output distribution.

$\mathbf{Hyb}_6$: In this hybrid, after the check on the servers' computation, if $\mathsf{CompCheck} = 0$, for each j with $\mathsf{Comp}_j = 0$, Sim additionally computes S_j's shares of $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$ from what $\mathcal{O}$ receives. Sim then checks whether these servers' shares of these sharings are all from valid $\Sigma/\Sigma^{(2)}/\Sigma^{(3)}$-sharings and valid $\Sigma^{(2)}$-sharings with all-zero secrets. If not, Sim sets $\mathsf{ShCheck} = 1$. This does not affect the output distribution. Thus, $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ have the same output distribution.

$\mathbf{Hyb}_7$: In this hybrid, for the input labels received from $\mathcal{F}_{\mathsf{OT}}$, if only for $\leq \kappa$ invocations $\mathcal{F}_{\mathsf{OT}}$, the labels of P_j's input shares received by $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{O}$ do not match, i.e., the labels received by $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{O}$ do not match among all $j \in \mathcal{V}$, then Sim changes the way of computing E's inputs to C. Sim first generates the $\leq \kappa$ shares containing these OT input bits whose labels are not consistent based on the shares for servers in $\mathcal{U}$ and then generates the other shares based on the generated shares and the real inputs of E. Otherwise, Sim first generates κ shares among the shares containing these OT input bits whose labels are not consistent based on the shares for servers in $\mathcal{U}$ and then generates the others. Since the OT input bits are randomly sampled Σ-sharings, the input bits are t-wise independent of the secrets. Thus, any $\leq \kappa$ shares of them can be generated without knowing the secrets based on the $t - \kappa$ servers' shares that have been generated in

Round 2. Therefore, we only change the order of generating different input shares without changing their distributions. Thus, $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have the same output distribution.

$\mathbf{Hyb}_8$: In this hybrid, for the input labels received from $\mathcal{F}_{\mathsf{OT}}$, if for more than κ invocations $\mathcal{F}_{\mathsf{OT}}$, the labels of P_j's input shares received by $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{O}$ do not match among all $j \in \mathcal{V}$, Sim does not follow the protocol to check them. Instead, Sim directly aborts the protocol on behalf of E in this case. This only changes the output distribution when the real check does not find the inconsistency of the labels in this case. Note that for those input wires with inconsistent labels, either of the labels can be found wrong if checked, which causes the abort of the evaluator. Since the Σ-sharing has the property that any t shares are uniformly random, the κ input shares generated first will cause the abort with at least $1 - 2^{-\kappa}$ probability. Therefore, the distribution only changes with probability $\leq 2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ are indistinguishable for $\mathcal{A}$.

$\mathbf{Hyb}_9$: In this hybrid, if the verification of E passes with $\mathsf{ShCheck} = 1$, Sim aborts the simulation. This only changes the distribution when there exists a set of $\leq t/2$ servers where each server S_j in them satisfies $\mathsf{Comp}_j = 1$, such that the other parties' shares of a Σ or $\Sigma^{(2)}$ or $\Sigma^{(3)}$-sharing or a $\Sigma^{(2)}$-sharing with an all-zero secret committed by G in the first round are not from a valid sharing, but they pass the check for the sharings. Note that $\geq n - t/2$ servers' shares of the sharings in VerSh must be computed from the committed shares, as $\mathsf{Comp} = 0$ for $\geq n - t/2$ parties. Assume the $\geq n - t/2$ servers' shares of a Σ-sharing $[\boldsymbol{x}_i]$ are not from a valid Σ-sharing, there is only one possible $\alpha_i \in \mathbb{F}_{2^{\kappa'}}$ that can make the sharings in VerSh valid when other coefficients are fixed. Otherwise, if there is an $\alpha_i' \neq \alpha_i$ that makes the $\geq n - t/2$ servers' shares of the sharings in VerSh valid, then the $n - t/2$ servers' shares of $(\alpha_i - \alpha_i') \cdot [\boldsymbol{x}_i]$ must comes from a valid Σ-sharing, which conflicts with the fact that the $\geq n - t/2$ servers' shares of $[\boldsymbol{x}_i]$ are invalid. If these servers' shares of a $\Sigma^{(2)}$-sharing or a $\Sigma^{(3)}$-sharing or a $\Sigma^{(2)}$-sharing of an all-zero secret are invalid, the case is similar. Therefore, if all coefficients are uniformly randomly sampled after the servers' views are committed, the output distribution only changes with probability $\leq 2^{-\kappa'}$ when the set of $n - t/2$ parties is fixed. Taking the union bound of $< 2^n$ all such sets of servers, the probability is still $\leq 2^{-(\kappa' - n)} = 2^{-\kappa}$.

Recall that the coefficients are determined by $\mathcal{O}(s')$, where s' is sampled randomly after the servers' views are committed. When $\mathcal{A}$ never queries s' before the servers' views are committed, the coefficients are uniformly random to him, and the distribution only changes with probability $\leq 2^{-\kappa}$. On the other hand, the probability that $\mathcal{A}$ queries $\mathcal{O}$ with s' before sending the commitments is no more than $T/2^{\kappa + \log T} = 2^{-\kappa}$, so the output distribution changes with $\leq 2^{-\kappa}$ in both cases. Thus, the distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ are indistinguishable for $\mathcal{A}$.

$\mathbf{Hyb}_{10}$: In this hybrid, if the verification of E passes with $\mathsf{CompCheck} = 1$, Sim aborts the simulation. This only changes the distribution when for $> t/2$ servers in the multiparty garbling process, G does not compute the garbled circuit from the committed views of them, or the labels of their output of $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^{G}$ received by $\mathcal{F}_{\mathsf{OT}}$ do not match the committed views. For all the cases where G does not compute the garbling of a server S_j from the committed view View^{S_j} except not using the committed labels as the inputs to $\mathcal{F}_{\mathsf{OT}}$, E can detect it if $j \in \mathcal{V}$. When $< 12\kappa$ servers' labels sent to $\mathcal{F}_{\mathsf{OT}}$ do not match the committed views in total, the distribution only changes when G does not compute the garbled circuit from the committed views of $> t/2 - 12\kappa > 70(\kappa + \log T)$ servers.

Recall that the set $\mathcal{V}$ is determined by $\mathcal{O}(\mathsf{GC} \| \mathsf{VerSh})$. That is to say, $\mathcal{V}$ is not determined before GC and VerSh are determined. Fixing GC and VerSh, the probability that $\mathcal{V}$ does not include the $> 18(\kappa + \log T)$ indices is no more than

$$\begin{aligned}
&\frac{n - 70(\kappa + \log T)}{n} \cdot \frac{n - 70(\kappa + \log T) - 1}{n - 1} \cdot \cdots \cdot \frac{n - 70(\kappa + \log T) - t/3 + 1}{n - t/3 + 1} \\
&< \left(\frac{n - 70(\kappa + \log T)}{n}\right)^{t/3} < \left(\frac{11}{12}\right)^{49(\kappa + \log T)} < 2^{-(\kappa + \log T)} = T^{-1} \cdot 2^{-\kappa}.
\end{aligned}$$

Note that $\mathcal{A}$ can query $\mathcal{O}$ for T times, where each time $\mathcal{A}$ can fix a pair of GC and VerSh and learn whether this pair can pass the check. Taking the union bound of them, the probability of $\mathcal{A}$ correctly obtaining a pair of GC and VerSh matching the condition that will make the output distribution change is no more than $2^{-\kappa}$. Otherwise, either the pair of GC and VerSh sent by G does not match the condition, or the query $\mathsf{GC} \| \mathsf{VerSh}$

is not made by the adversary $\mathcal{A}$. If the query is not made, $\mathcal{V}$ is random to $\mathcal{A}$, and the probability that the distribution changes is no more than $T^{-1} \cdot 2^{-\kappa}$. In conclusion, the output distribution only changes with probability of $\Theta(1)/2^{\kappa}$.

On the other hand, note that when $> \kappa$ parties' labels sent to $\mathcal{F}_{\mathsf{OT}}$ among the input labels for S_j with $k \in \mathcal{V}$ do not match the committed views, Sim will directly abort the protocol on behalf of E. Consider X_j to be 1 when S_j's committed labels match what G sent to $\mathcal{F}_{\mathsf{OT}}$ and $S_j \in \mathcal{V}$. When $e \geq 12\kappa$ pairs of labels sent to $\mathcal{F}_{\mathsf{OT}}$ do not match the committed views in total, if $\mathcal{V}$ is random, $X_1, \ldots, X_n$ are independent random variables. Let $X = X_1 + \cdots + X_e$, then the expectation of X is $\mu = e/3 \geq 4\kappa$, and Sim will not directly abort the protocol because of this reason only when $X < \kappa$. The probability is

$$P(X < \kappa) = P(1 - 3\mu/4) \leq e^{-3^2\kappa/(2\cdot 4)} < 2^{-\kappa}$$

by Chernoff bound. For the same reason, the output distribution only changes with probability of $\Theta(1)/2^{\kappa} + 2^{-\kappa}$.

Thus, the distributions of $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ are indistinguishable for $\mathcal{A}$.

$\mathbf{Hyb}_{11}$: In this hybrid, Sim does not follow the protocol to evaluate the garbled circuit on behalf of E. Instead, Sim only follows that protocol to evaluate the garbled subcircuits for the $\geq n - t$ servers S_j with $\mathsf{Comp}_j = 0$ whose shares of the $\Sigma, \Sigma^{(2)}, \Sigma^{(3)}$-sharings and the $\Sigma^{(2)}$-sharings with all-zero secrets are all from valid sharings. The secrets of $\Sigma^{(2)}, \Sigma^{(3)}$-sharings required to reconstruct in the evaluation process are reconstructed from the shares for these servers. Note that from the correctness of our garbling scheme, these servers' shares of the $\Sigma^{(2)}, \Sigma^{(3)}$-sharings reconstructed in the evaluation process are valid. The only difference from the original evaluation process is that Sim does not evaluate the sub-circuits for the other servers, and he does not use error correction to reconstruct the secrets for these sharings. Since the $\Sigma^{(2)}, \Sigma^{(3)}$-sharings have error-correction properties for up to t errors, no matter how the other t servers' garbled sub-circuits are evaluated, the secrets of these sharings are fixed, so we do not change any message sent to G. Thus, $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ have the same output distribution.

$\mathbf{Hyb}_{12}$: In this hybrid, Sim additionally computes the valid shares and garbled sub-circuits for the $\leq t$ servers, excluding the $\geq n - t$ servers taken in the last hybrid, from the committed views to them and simulates the whole evaluation process honestly using these garbled sub-circuits. Since the secrets of the $\Sigma^{(2)}, \Sigma^{(3)}$-sharings reconstructed in the evaluation process have already been determined by the other $n - t$ parties, we do not change any message sent to G. Thus, $\mathbf{Hyb}_{12}$ and $\mathbf{Hyb}_{11}$ have the same output distribution.

$\mathbf{Hyb}_{13}$: In this hybrid, Sim does not follow the protocol to evaluate the circuit and sends the garbled output $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{R_i = G}$ to G on behalf of E. Instead, Sim extracts the output of $\mathcal{F}_{\mathsf{input}}^{G}$ from the commitments generated by G and reconstructs the inputs of G together with $\mathsf{mask}, \boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$. Sim then sends the inputs of G to $\mathcal{F}$ and receives G's output. Then, Sim uses $\mathsf{mask}, \boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$ and the circuit outputs of G to compute G's outputs of $\overline{C}$, i.e., $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$. Sim then computes the corresponding $\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)} = \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)} + (\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}) * \boldsymbol{s}_i$ for each $\alpha = 1, \ldots, \log 40T$, where $\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}, \boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)}$ are reconstructed from the sharings $[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]$ received by $\mathcal{O}$. In addition, E does not compute his output by himself. Instead, he directly gets his outputs from $\mathcal{F}$. Note that the shares of the input sharings generated by E for the $\geq n - t$ servers taken in $\mathbf{Hyb}_{13}$ and their corresponding wire labels are not used for the simulation. Sim does not generate them in this hybrid.

Since the garbling and evaluation in $\mathbf{Hyb}_{12}$ completely follow the protocol, from the correctness of the garbling scheme, the outputs of E computed in the two hybrids are the same. If G follows the protocol to compute his output, he will also get the correct output. Note that in the explicit protocol of Π_0, when $\mathsf{mask}, \boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$ are fixed, the computation from $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ to G's output is deterministic and invertible. Furthermore, when $\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}, \boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)}$ are fixed by the commitments, $\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}$ is also determined by $\boldsymbol{s}_i$. Therefore, we only change the order of generating G's output and the messages $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ sent to G without changing their distributions. Thus, $\mathbf{Hyb}_{13}$ and $\mathbf{Hyb}_{12}$ have the same output distribution.

Note that $\mathbf{Hyb}_{13}$ is the ideal-world scenario, where the statistical difference between any two hybrids $\mathbf{Hyb}_i$ and $\mathbf{Hyb}_{i+1}$ $(i = 0, \ldots, 12)$ is no more than $\Theta(1)/2^{\kappa}$. Thus, the total difference between $\mathbf{Hyb}_0$ and

$\mathbf{Hyb}_{13}$ is still at most $\Theta(1)/2^\kappa$, so Π securely realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model against $\mathcal{A}$ when the garbler G is corrupted.

I.2 Security against a Corrupted Evaluator

When E is corrupted, we construct the ideal adversary Sim below. Similarly, we separate the description of Sim into the simulators $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3, \mathsf{Sim}_4$ for the 4 rounds.

Simulator Sim_1

1. Sim follows the protocol to do the initialization and get the protocol Π'_0. Then, Sim sets $\mathcal{C} = \emptyset$, $\mathsf{ShCheck} = 0$, $\mathsf{CompCheck} = 0$, and $\mathsf{Comp}_j = \mathsf{Corr}_j = 0$ for each $j = 1, \ldots, n$.
2. Sim runs Steps 3-4 and Steps 6-8 of $\mathsf{Sim}_{\mathsf{pri}}$ (similar to $\mathsf{Sim}_{\mathsf{obl}}$) and obtains the garbled circuit $\tilde{C}$. In particular, Sim does not generate d_{mac}. Sim still honestly emulates $\mathcal{O}$.
3. For each bit b (indexed i) of the sharing $[\boldsymbol{r}_0]_{\times\kappa'}$ (where b will be received from E later), Sim samples a random label $L_{i,b} \in \mathbb{F}_2^{\log 40T}$. Then, if b is from the share of a server S_j, Sim adds $L_{i,b} \oplus \Delta^{S_j}$ into Q_j.
4. From this step, once $\mathcal{A}$ queries a random oracle with query q:
   - If $q \notin Q_j$ for all $j \in \{1, \ldots, n\}$, Sim does nothing.
   - If q is in some Q_j, then Sim sets $\mathsf{Corr}_j = 1$ and adds S_j into $\mathcal{C}$. If $|\mathcal{C}| > 2t/3$ with $\mathsf{CompCheck} = \mathsf{ShCheck} = 0$, Sim aborts the simulation.
5. Sim emulates $\mathcal{F}_{\mathsf{OT}}$ to receive the input sharings $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ from E.
6. For each wire w whose value is a bit of their input shares, Sim emulates $\mathcal{F}_{\mathsf{OT}}$ to send $k_{w,v_w \oplus \lambda_w} \| (v_w \oplus \lambda_w)$ to E, where $k_{w,v_w \oplus \lambda_w}, v_w \oplus \lambda_w$ are sampled while generating $\tilde{C}$.
7. Sim emulates $\mathcal{F}_{\mathsf{OT}}$ to receive the mask sharing $[\boldsymbol{r}_0]_{\times\kappa'}$ from E. Then for each bit b (indexed i) of the sharing, Sim emulates $\mathcal{F}_{\mathsf{OT}}$ to send $L_{i,b} \in \mathbb{F}_2^{\log 40T}$ to E.
8. Sim samples a random $(\kappa + \log T)$-bit string as s and emulates $\mathcal{F}_{\mathsf{OT}}$ to send it to E.
9. Sim generates n random strings as $c_1, \ldots, c_n$ and $n(n-1)$ random sharings as $\sigma_{i,j}$ for all pairs of servers (S_i, S_j), where the length of the strings matches the output length of $\mathcal{O}$ and emulates $\mathcal{F}_{\mathsf{OT}}$ to send them to E.

Figure 24: The simulator for the first round of Π when E is corrupted.

Simulator Sim_2

1. Sim receives $[\boldsymbol{v}]_{\times\kappa'}, \mathcal{U}, s'$ from E on behalf of G.
2. Sim checks whether each S_j's share of $[\boldsymbol{v}]_{\times\kappa'}$ is computed honestly from S_j's shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$ (received from $\mathcal{F}_{\mathsf{OT}}$). If not, Sim sets $\mathsf{Comp}_j = 1$ and includes S_j into $\mathcal{C}$. Then:
   - If $|\mathcal{C}| \leq 2t/3$, then Sim runs Sim_0 to compute S_j's view based on the views other parties in $\mathcal{C}$. For each $i = 1, \ldots, \mathsf{rec} + n_o$, Sim samples S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for the i-th reconstruction based on the shares for servers in $\mathcal{C}$ and computes S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ accordingly by:
   $$[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)} = [\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)} \oplus [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}] \otimes [\boldsymbol{s}_i]^{(2)}.$$
   - If $|\mathcal{C}| > 2t/3$ and there are only $\leq t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim aborts the simulation.

   If there are $> t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim sets $\mathsf{CompCheck} = 1$.
3. If $\mathsf{CompCheck} = 0$, for each j with $\mathsf{Comp}_j = 0$, Sim obtains S_j's shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ from what E sends to $\mathcal{F}_{\mathsf{OT}}$ in the first round. Sim then checks whether these servers' shares of these sharings are all from valid Σ-sharings.
   - If not, Sim sets $\mathsf{ShCheck} = 1$.

– Otherwise, Sim reconstructs the secrets, i.e., E's inputs to C, from these sharings. Sim then sends E's inputs to $\mathcal{F}$ and learns E's outputs of C, which is equal to E's output of $\overline{C}$.

Figure 25: The simulator for the second round of Π when E is corrupted.

Simulator Sim_3

1. Sim follows the protocol to perform the check on the evaluator's input sharings. If the check fails, Sim aborts the protocol on behalf of G.
2. If $\mathsf{ShCheck} = 1$ or $\mathsf{CompCheck} = 1$ but the check passes, Sim aborts the simulation.
3. Sim generates a random $\Sigma_{\times\kappa'}$-sharing, a random $\Sigma^{(2)}_{\times\kappa'}$-sharing, and a random $\Sigma^{(3)}_{\times\kappa'}$-sharing as the three sharings in VerSh.
4. For each $S_j \in \mathcal{C}$, Sim samples all the wire masks for $\mathtt{Circ}^{S_j}$ excluding the wire masks for output wires of XOR/output gates. Sim also samples the shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for parties in $\mathcal{C}$. Then, Sim honestly emulates $\mathcal{O}$ to compute all the outputs for queries in Q_j for all $S_j \in \mathcal{C}$, and then Sim updates the ciphertexts in $\{F_j\}_{S_j \in \mathcal{C}}$ and F_{ct} using these query outputs. In particular, for each ciphertext of the form $\mathcal{O}(q) \oplus \mathsf{m}$ for some $q \in Q_j$ with $S_j \in \mathcal{C}$, m can be obtained by honestly running the garbling process for S_j, and then the ciphertext can be computed from $\mathcal{O}(q)$ and m. Now Sim has already got GC and Versh.
5. Sim honestly emulates $\mathcal{O}$ to compute $\mathcal{O}(\mathsf{GC}\|\mathsf{Versh})$ and get $\mathcal{V}$. For each $j \in \mathcal{V}$, Sim sets $\mathsf{Corr}_j = 1$ and includes S_j into $\mathcal{C}$. Then:
   – If $|\mathcal{C}| \leq t$, then Sim runs Sim'_0 to compute S_j's view based on the views other parties in $\mathcal{C}$. In particular, E is set to be corrupted. E's output from of $\overline{C}$ is set to be his output from $\mathcal{F}$. While emulating $\mathcal{F}^E_{\mathsf{input}}$, the outputs are set to be the input sharings received from $\mathcal{F}_{\mathsf{OT}}$ after error correction. For each $i = 1, \ldots, \mathsf{rec} + n_o$, Sim samples S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for the i-th reconstruction based on the shares for servers in $\mathcal{C}$ and computes S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ accordingly by:
   $$[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)} = [\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)} \oplus [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}] \otimes [\boldsymbol{s}_i]^{(2)}.$$
   – If $|\mathcal{C}| > t$, Sim aborts the simulation.

   From this step, Sim follows Step 9 of $\mathsf{Sim}_{\mathsf{pri}}$ to emulate $\mathcal{O}$. More concretely, once $\mathcal{A}$ queries a random oracle with query q:
   – If $q \notin Q_j$ for all $j \in \{1, \ldots, n\}$, $\mathsf{Sim}_{\mathsf{pri}}$ honestly emulates the random oracle.
   – If q is in some Q_j, then if $S_j \notin \mathcal{C}$, $\mathsf{Sim}_{\mathsf{pri}}$ sets $\mathsf{Corr}_j = 1$ and adds S_j into $\mathcal{C}$. Then:
   * If $|\mathcal{C}| > t$, $\mathsf{Sim}_{\mathsf{pri}}$ aborts the simulation.
   * Otherwise, $\mathsf{Sim}_{\mathsf{pri}}$ runs Sim'_0 to compute S_j's view suring Π'_0 based on the views other parties in $\mathcal{C}$. Then, for each $i = 1, \ldots, \mathsf{rec} + n_o$, $\mathsf{Sim}_{\mathsf{pri}}$ samples S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for the i-th reconstruction based on the shares for servers in $\mathcal{C}$ and computes S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ accordingly by:
   $$[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)} = [\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)} \oplus [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}] \otimes [\boldsymbol{s}_i]^{(2)}.$$

   Then the response is determined via the following process:
   * If the queried oracle is $\mathcal{O}_1$, the response to q is determined by the equation for computing a ciphertext, which is of the form
   $$\mathsf{ct} = \mathcal{O}_1(q) \oplus \mathsf{m}$$
   for a message m that can be computed from S_j's view and the sampled wire labels. Note that all the ciphertexts have already been sampled. Then, the response can be computed by $\mathsf{ct} \oplus m$. The remaining bits of the output from $\mathcal{O}$ for the same query are then randomly sampled. Then the output of $\mathcal{O}$ for this query is fixed.
   * If the queried oracle is $\mathcal{O}_2$, the response to q is also determined by the equation for computing a ciphertext, which is of the form
   $$\mathsf{ct} = \mathcal{O}_2(q) \oplus \mathsf{m}$$

for a message m that can be computed from S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$. Note that all the ciphertexts have already been sampled. Then, the response can also be computed by $\mathsf{ct} \oplus m$. The remaining bits of the output from $\mathcal{O}$ for the same query are then randomly sampled. Then the output of $\mathcal{O}$ for this query is fixed.

6. For each $S_j \in \mathcal{C}$, Sim computes S_j's shares of $[\boldsymbol{r}_1]_{\times\kappa'}, [\boldsymbol{r}_2]^{(2)}_{\times\kappa'}, [\boldsymbol{r}_3]^{(3)}_{\times\kappa'}, [\boldsymbol{o}]^{(2)}_{\times\kappa'}$ from:
   - The four sharings in VerSh;
   - The coefficients $\{\alpha_i\}_{i=1}^{k_1}, \{\beta_i\}_{i=1}^{k_2}, \{\gamma_i\}_{i=1}^{k_3}, \{\eta_i\}_{i=1}^{k_4}$ obtained from $\mathcal{O}(s')$;
   - S_j's shares of $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, and $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$ obtained from the view of S_j during the execution of Π_0' and S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}], [\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ that has been generated.

   Now, Sim has obtained $\{\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}\}_{j\in\mathcal{V}}$.

7. For each $j \in \mathcal{V}$, Sim samples a random $(\kappa + \log T)$-bit string as $\boldsymbol{r}_{\mathsf{commit}}^{S_j}$. If $\mathsf{View}^{S_j} \| \boldsymbol{r}_{\mathsf{commit}}^{S_j} \| j$ has been queried by $\mathcal{A}$, then Sim aborts the simulation. Otherwise, Sim sets the output of $\mathcal{O}(\mathsf{View}^{S_j} \| \boldsymbol{r}_{\mathsf{commit}}^{S_j} \| j)$ to c_j. Similarly, for each $j \in \mathcal{V}$ and $i > j$ or $i \notin \mathcal{V}$, Sim samples 2 random $(\kappa + \log T)$-bit strings as $\boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}, \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}$. If $\mathsf{Sh}_{i,j} \| \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)} \| i \| j$ or $\mathsf{Sh}_{j,i} \| \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)} \| j \| i$ has been queried by $\mathcal{A}$, then Sim aborts the simulation. Otherwise, Sim sets the output of $\mathcal{O}(\mathsf{Sh}_{i,j} \| \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)} \| i \| j)$ and $\mathsf{Sh}_{j,i} \| \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)} \| j \| i$ to $\sigma_{i,j}$ and $\sigma_{j,i}$ respectively. Finally, Sim sends $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}\}_{j\in\mathcal{V}}, \{\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}, \mathsf{Sh}_{j,i}, \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}\}_{i\in\mathcal{V}, j\in\{1,\ldots,n\}\setminus\{i\}})$ to E on behalf of G.

Figure 26: The simulator for the third round of Π when E is corrupted.

Simulator Sim_4

1. Sim receives $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ from E on behalf of G.
2. Sim checks whether $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ matches what Sim generates. If not, Sim aborts the protocol on behalf of G.
3. Sim outputs what $\mathcal{A}$ outputs.

Figure 27: The simulator for the fourth round of Π when E is corrupted.

We construct the following hybrids:

$\mathbf{Hyb}_0$: In this hybrid, Sim gets G's inputs and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, Sim additionally sets $\mathcal{C} = \emptyset$, $\mathsf{ShCheck} = 0$, $\mathsf{CompCheck} = 0$, and $\mathsf{Comp}_j = \mathsf{Corr}_j = 0$ for each $j = 1, \ldots, n$ at the beginning of the simulation. This does not affect the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, Sim follows $\mathsf{Sim}_{\mathsf{pri}}$ to generate the sets $Q_1, \ldots, Q_n$ at the beginning, once $\mathcal{A}$ queries a random oracle with query q:

- If $q \notin Q_j$ for all $j \in \{1, \ldots, n\}$, Sim does nothing.
- If q is in some Q_j, then Sim sets $\mathsf{Corr}_j = 1$ and adds S_j into $\mathcal{C}$. If $|\mathcal{C}| > 2t/3$ with $\mathsf{CompCheck} = \mathsf{ShCheck} = 0$, Sim aborts the simulation.

This only changes the output distribution when $\mathcal{A}$ queries a query in $> 2t/3 > \kappa$ sets among $Q_1, \ldots, Q_n$. As argued in the proof of privacy for our garbling scheme, this only happens with $< 2^{-\kappa}$ probability. Thus, the distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are indistinguishable for $\mathcal{A}$.

$\mathbf{Hyb}_3$: In this hybrid, Sim does not follow the protocol to generate the wire labels for the sharings $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$. Instead, for each wire w whose wire value is from S_j's shares of these sharings, Sim first samples wire label $k_{w,\lambda_w\oplus b_w} \| (\lambda_w \oplus b_w) \in \mathbb{F}_2^{\log 40T+1}$ randomly, where b_w is the wire value $\mathcal{F}_{\mathsf{OT}}$ receives from E. Then, Sim computes the other label by $k_{w,\lambda_w\oplus b_w\oplus} = k_{w,\lambda_w\oplus b_w} \oplus \Delta^{S_j}$ and computes the wire masks by $\lambda_w = (\lambda_w \oplus b_w) \oplus b_w$. For each bit b (indexed i) of the sharing $[\boldsymbol{r}_0]_{\times\kappa'}$ (where b will be received from E

later), Sim has already sampled $L_{i,b}$. Then, if b is from the share of a server S_j, Sim adds $L_{i,b} \oplus \Delta^{S_j}$ into Q_j, which does not change the output distribution. Since $k_{w,0}$ and λ_w are uniformly randomly distributed, so is $k_{w,\lambda_w \oplus b_w} \| (\lambda_w \oplus b_w)$. We only change the order of sampling:

- λ_w and $\lambda_w \oplus b_w$;
- $k_{w,0}$ and $k_{w,1}$ when $\lambda_w \oplus b_w = 1$

without changing their distributions. Thus, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have the same output distribution.

$\mathbf{Hyb}_4$: In this hybrid, Sim does not follow the protocol to compute $c_1, \dots, c_n$ and $\sigma_{i,j}$ for each pair of servers (S_i, S_j). Instead, Sim directly samples random strings (with the length as an RO output) as them. Then, the corresponding RO outputs are set to these values. In addition, the generations of $\boldsymbol{r}_{\mathsf{commit}}^{S_1}, \dots, \boldsymbol{r}_{\mathsf{commit}}^{S_n}$ and $\boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}$ for each pair of servers (S_i, S_j) are delayed to Round 3. When they are required before Round 3, Sim aborts the simulation.

Note that if $\mathcal{A}$ has not queried $\mathcal{O}$ before they are sent, we do not make any changes. Therefore, the output distribution only changes when one of the queries has been made by $\mathcal{A}$. However, each of the queries contains a uniformly random $(\kappa + \log T)$-bit string, and it is not used to compute any message sent to $\mathcal{A}$ before sending the commitments. Therefore, the probability that the query has been made is no more than $T/2^{\kappa+\log T} = 2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ are indistinguishable for $\mathcal{A}$.

Now, the sharings $[\boldsymbol{r}_1]_{\times\kappa'}, [\boldsymbol{r}_2]^{(2)}_{\times\kappa'}, [\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$, and $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ are not used during the simulation until Round 3, so Sim delays the generations of them into Round 3.

$\mathbf{Hyb}_5$: In this hybrid, after receiving $[\boldsymbol{v}]_{\times\kappa'}, \mathcal{U}, s'$ from E, Sim additionally checks whether each S_j's share of $[\boldsymbol{v}]_{\times\kappa'}$ is computed honestly from S_j's shares of $[\boldsymbol{I}_1], \dots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$ (received from $\mathcal{F}_{\mathsf{OT}}$). If not, Sim sets $\mathsf{Comp}_j = 1$ and includes S_j into $\mathcal{C}$. If there are $\geq t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim sets $\mathsf{CompCheck} = 1$. This doesn't affect the output distribution. Thus, $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have the same output distribution.

$\mathbf{Hyb}_6$: In this hybrid, after the check added in the last hybrid, if $|\mathcal{C}| > 2t/3$ and there are only $\leq t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim aborts the simulation. Note that a server S_j with $\mathsf{Comp}_j \neq 1$ is added into $\mathcal{C}$ only because a query in Q_j has been queried. Note that until this step, Sim hasn't used the queries in Q_j to generate any messages sent to E before one of them is queried if $\mathsf{Comp}_j = 0$, which is the same case as in $\mathbf{Hyb}_2$. For the same reason, this happens with $< 2^{-\kappa}$ probability. Thus, the distributions of $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ are indistinguishable for $\mathcal{A}$.

$\mathbf{Hyb}_7$: In this hybrid, after the check on the servers' computation, if $\mathsf{CompCheck} = 0$, for each j with $\mathsf{Comp}_j = 0$, Sim additionally computes S_j's shares of $[\boldsymbol{I}_1], \dots, [\boldsymbol{I}_{k_0}]$ from what $\mathcal{F}_{\mathsf{OT}}$ receives. Sim then checks whether these servers' shares of these sharings are all from valid Σ-sharings. If not, Sim sets $\mathsf{ShCheck} = 1$. This does not affect the output distribution. Thus, $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have the same output distribution.

$\mathbf{Hyb}_8$: In this hybrid, if the verification of E passes with $\mathsf{ShCheck} = 1$, Sim aborts the simulation. This only changes the distribution when there exists a set of $\leq t/3$ servers where each server S_j in them satisfies $\mathsf{Comp}_j = 1$, such that the other parties' shares of an input Σ-sharing committed by E in the first round are not from a valid sharing, but they pass the check for the sharings. Note that $\geq n - t/3$ servers' shares of the sharing $[\boldsymbol{v}]_{\times\kappa'}$ must be computed from the shares sent to $\mathcal{F}_{\mathsf{OT}}$, as $\mathsf{Comp} = 0$ for $\geq n - t/3$ parties. Assume the $\geq n - t/3$ servers' shares of a Σ-sharing $[\boldsymbol{I}_i]$ are not from a valid Σ-sharing, there is only one possible $\alpha_i \in \mathbb{F}_{2^{\kappa'}}$ that can make the sharings in VerSh valid when other coefficients are fixed. Otherwise, if there is an $\alpha_i' \neq \alpha_i$ that makes the $\geq n - t/3$ servers' shares of the sharing $[\boldsymbol{v}]_{\times\kappa'}$ valid, then the $n - t/3$ servers' shares of $(\alpha_i - \alpha_i') \cdot [\boldsymbol{I}_i]$ must comes from a valid Σ-sharing, which conflicts with the fact that the $\geq n - t/3$ servers' shares of $[\boldsymbol{I}_i]$ are invalid. Therefore, if all coefficients are uniformly randomly sampled after the servers' views are committed, the output distribution only changes with probability $\leq 2^{-\kappa'}$ when the set of $n - t/3$ parties is fixed. Taking the union bound of $< 2^n$ all such sets of servers, the probability is still $\leq 2^{-(\kappa'-n)} = 2^{-\kappa}$.

Recall that the coefficients are determined by $\mathcal{O}(s)$, where s is sampled randomly after the servers' views are committed. When $\mathcal{A}$ never queries s before the servers' views are committed, the coefficients are uniformly random to him, and the distribution only changes with probability $\leq 2^{-\kappa}$. On the other hand, the probability that $\mathcal{A}$ queries $\mathcal{O}$ with s before sending the commitments is no more than $T/2^{\kappa+\log T} = 2^{-\kappa}$,

so the output distribution changes with $\leq 2^{-\kappa}$ in both cases. Thus, the distributions of $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ are indistinguishable for $\mathcal{A}$.

$\mathbf{Hyb}_9$: In this hybrid, after checking the evaluator's input sharings in Round 3, Sim aborts the simulation if the check passes with $\mathsf{CompCheck} = 1$. Recall that $\mathsf{CompCheck} = 1$ only when more than $t/3$ servers S_j's shares of $[\boldsymbol{v}]_{\times\kappa'}$ are not computed honestly from their shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$ with $\mathsf{Corr}_j = 0$. If a server S_j among these servers is included in $\mathcal{U}$, then the adversary should be able to open one of the labels that E does not get. Otherwise, E cannot pass the check, and the output distribution won't change.

Note that for $[\boldsymbol{r}_0]_{\times\kappa'}$, the other label for each bit of this sharing is not generated in the first 2 rounds. There is no more than $2^{-\log 40T} = 1/40T$ probability that the other label opened by $\mathcal{A}$ matches what is sampled in Round 3. For each bit of the input sharings $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$, the XOR of the other label $L_{i,1\oplus b}$ (for the i-th bit b of $[\boldsymbol{r}_0]_{\times\kappa'}$) and the label $L_{i,b}$ E obtained from $\mathcal{F}_{\mathsf{OT}}$ is Δ^{S_j} if the bit is in a share for S_j. However, Δ^{S_j} is only used for computing a message to E when a query in Q_j is called, which will lead to $\mathsf{Corr}_j = 1$.

Recall that $\mathsf{Comp}_j = 1$ only happens when $[\boldsymbol{v}]_{\times\kappa'}$ is received with $\mathsf{Corr}_j = 0$, which shows that Δ^{S_j} is not used in the computation of any message to $\mathcal{A}$. Therefore, there is no more than $2^{-\log 40T} = 1/40T$ probability that the other label opened by $\mathcal{A}$ matches Δ^{S_j} sampled in Round 3. Thus, for each fixed j, if $S_j \in \mathcal{U}$ and $\mathsf{Comp}_j = 1$, E can only pass the check on E's labels for no more than $1/40T$ probability. Since the sampling processes of each Δ^{S_j} are independent, the check can pass with no more than $T^{-\kappa}$ probability when at least κ servers S_j with $\mathsf{Comp}_j = 1$ and $\mathsf{Corr}_j = 0$ are included in $\mathcal{U}$.

Therefore, if $\mathcal{U}$ is uniformly random and sampled after $[\boldsymbol{v}]_{\times\kappa'}$ is sent, then the probability that κ servers S_j among the $t/3$ servers (whose shares of $[\boldsymbol{v}]_{\times\kappa'}$ are not computed honestly from their shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$) are not included in $\mathcal{U}$ can be estimated by the Chernoff bound. More concretely, we add the servers into $\mathcal{U}$ one by one. For each choice of server in $\mathcal{U}$, if the current number of servers $S_j \in \mathcal{U}$ with $\mathsf{Comp}_j = 1$ is smaller than κ, then the probability that this choice of server is among these servers is at least $(t/3 - \kappa)/n$. Note that the probability that κ servers S_j among the $t/3$ servers are chosen in $\mathcal{U}$ is smaller than the probability if each time a random server (allowing repeated choices) is added into $\mathcal{U}$. In this case, each time a server is added, let i be the number of servers in $\mathcal{U}$ after adding this server, then we define:

- $X_i = 1$ if the chosen party is not among a randomly chosen set of $t/3 - \kappa$ servers that are not included in $\mathcal{U}$ before.
- $X_i = 0$ otherwise.

Then, $X_1, \ldots, X_{t/3}$ are independent, with the probability of each $X_i = 1$ be $(t/3 - \kappa)/n$. Let $X = X_1 + \cdots + X_{t/3}$. Then, $\mu := \mathbb{E}(X) = t(t/3 - \kappa)/3n > 4(\kappa + \kappa + \log T)$, so by Chernoff Bound:

$$P(X \leq \kappa) \leq P(X \leq (1 - 3/4)\mu) \leq e^{-\frac{9}{32}\cdot\mu} \leq 2^{-\kappa} \cdot 2^{-\kappa} \cdot T^{-1}.$$

Thus, if $\mathcal{U}$ is uniformly random, the output distribution only changes with probability $1 - (1 - 2^{-\kappa} \cdot 2^{-\kappa} \cdot T^{-1}) \cdot (1 - T^{-\kappa}) < 2^{-\kappa} \cdot T^{-1}$.

Note that when $[\boldsymbol{v}]_{\times\kappa'}$ is not queried by $\mathcal{A}$ to $\mathcal{O}$, then $\mathcal{U}$ is indeed uniformly random. In this case, the distribution only changes with no more than $2^{-\kappa} \cdot T^{-1}$ probability. On the other hand, $\mathcal{A}$ has at most T queries to $\mathcal{O}$, where each time $\mathcal{A}$ needs to fix $[\boldsymbol{v}]_{\times\kappa'}$ and learn whether it can pass the check, so each time $[\boldsymbol{v}]_{\times\kappa'}$ only passes the check with no more than $2^{-\kappa} \cdot T^{-1}$ probability. Taking the union bound of T queries, the probability that there exists a query of the form $[\boldsymbol{v}]_{\times\kappa'}$ by $\mathcal{A}$ that can pass the check (with more than $t/3$ servers S_j' shares of $[\boldsymbol{v}]_{\times\kappa'}$ are not computed honestly from their shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$ with $\mathsf{Corr}_j = 0$) is no more than $T \cdot 2^{-\kappa} \cdot T^{-1} = 2^{-\kappa}$. If it does not exist, $\mathcal{A}$ either sends a query he made before which would not change the distribution, or he will send another sharing $[\boldsymbol{v}]_{\times\kappa'}$ that he has not queried, which changes the output distribution with no more than $2^{-\kappa}$ probability. Therefore, the total probability that the output distribution changes is no more than $2^{-\kappa} + (1 - 2^{-\kappa}) \cdot 2^{-\kappa} = \Theta(1)/2^{\kappa}$. Thus, the distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ are indistinguishable for $\mathcal{A}$.

$\mathbf{Hyb}_{10}$: In this hybrid, Sim does not follow the protocol to sample $[\boldsymbol{r}_1]_{\times\kappa'}, [\boldsymbol{r}_2]^{(2)}_{\times\kappa'}, [\boldsymbol{r}_3]^{(3)}_{\times\kappa'}, [\boldsymbol{o}]^{(2)}_{\times\kappa'}$ and compute the 4 sharings in VerSh. Instead, Sim samples the 4 sharings in VerSh randomly and then computes $[\boldsymbol{r}_1]_{\times\kappa'}, [\boldsymbol{r}_2]^{(2)}_{\times\kappa'}, [\boldsymbol{r}_3]^{(3)}_{\times\kappa'}, [\boldsymbol{o}]^{(2)}_{\times\kappa'}$ based on VerSh, the coefficients computed from $\mathcal{O}(s')$, $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, and $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$. Since $[\boldsymbol{r}_1]_{\times\kappa'}, [\boldsymbol{r}_2]^{(2)}_{\times\kappa'}, [\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$ are uniformly random $\Sigma_{\times\kappa'}, \Sigma^{(2)}_{\times\kappa'}, \Sigma^{(3)}_{\times\kappa'}$-sharings, and $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ is a uniformly random $\Sigma^{(2)}$-sharing with an all-zero secret, so are the corresponding sharings in VerSh. Therefore, we only change the order of them without changing their distributions. Thus, $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ have the same output distribution.

$\mathbf{Hyb}_{11}$: In this hybrid, Sim does not follow the protocol to generate the garbled circuit. Instead, for each server S_j that is not in the set $\mathcal{C}$ after performing the verification and sampling VerSh in Round 3, Sim does not follow the protocol to query the random oracle with the queries in Q_j first and then compute the ciphertexts in $\tilde{C}$ with these outputs. Instead, Sim follows $\mathsf{Sim}_{\mathsf{obl}}$ to generate these ciphertexts and emulate $\mathcal{O}$ in the future simulation. Following from the proof of privacy of our garbling scheme, the generated $\tilde{C}$ has the same distribution as computed by following the protocol, and the emulation of $\mathcal{O}$ can be programmed when $\mathcal{A}$ queries an element in Q_j for no more than t $j \in \{1, \ldots, n\}$. Note that the queries in Q_j are not used for sending any message to $\mathcal{A}$ if $S_j \notin \mathcal{V}$ and $\mathsf{Comp}_j = 0$. The distribution only changes when $\mathcal{A}$ queries $t/3$ of them. For the same reason as in $\mathbf{Hyb}_2$, this happens with at most $2^{-\kappa}$ probability. Thus, the distributions of $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ are indistinguishable for $\mathcal{A}$.

Note that for the servers that are never included in $\mathcal{C}$, Sim does not need their shares of $[\boldsymbol{r}_1]_{\times\kappa'}$, $[\boldsymbol{r}_2]^{(2)}_{\times\kappa'}$, $[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$, and $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ in the simulation. Sim does not generate them in the future hybrids. In addition, Sim does not need the set $\{\boldsymbol{r}^{(1)}_{\boldsymbol{s}_i\oplus 1}, \ldots, \boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}_i\oplus 1}\}^{\mathsf{rec}+n_o}_{i=\mathsf{rec}+1}$ during the simulation, Sim also does not generate them in future hybrids.

$\mathbf{Hyb}_{12}$: In this hybrid, Sim does not sample $\boldsymbol{r}^{S_j}_{\mathsf{commit}}$ and $\boldsymbol{r}^{(i,j)}_{\mathsf{comsh}}$ if $i, j \notin \mathcal{V}$ and check whether the corresponding query to the random oracle has been queried. Note that Sim does not need these random strings except that Sim checks whether these queries are made. Therefore, this only changes the output distribution when one of them is truly queried. Since they are not used for sending any messages to $\mathcal{A}$, each query made by $\mathcal{A}$ only has no more than $1/2^{\log T+\kappa} = T^{-1} \cdot 2^{-\kappa}$ probability to match any of the queries. Taking the union bound for all T possible queries made by $\mathcal{A}$, the probability is still $\leq 2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_{12}$ and $\mathbf{Hyb}_{11}$ are indistinguishable for $\mathcal{A}$.

$\mathbf{Hyb}_{13}$: In this hybrid, Sim does not follow the protocol to check the garbled outputs received from the evaluator. Instead, Sim directly checks whether $\{\boldsymbol{s}_i, \boldsymbol{r}^{(1)}_{\boldsymbol{s}_i}, \ldots, \boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}_i}\}^{\mathsf{rec}+n_o}_{i=\mathsf{rec}+1}$ matches what Sim generates. As analyzed in the proof of authenticity of our garbling scheme, this only changes the output distribution when E outputs $Y = (\kappa, \{\boldsymbol{s}'_i, \boldsymbol{r}^{(1)}_{\boldsymbol{s}'_i} \ldots, \boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}'_i}\}^{\mathsf{rec}+n_o}_{i=\mathsf{rec}+1})$ such that the check passes, but $\{\boldsymbol{s}'_i\}^{\mathsf{rec}+n_o}_{i=\mathsf{rec}+1}$ flips $t+1$ bits from $\{\boldsymbol{s}_i\}^{\mathsf{rec}+n_o}_{i=\mathsf{rec}+1}$. Since Sim does not generate $\{\boldsymbol{r}^{(\alpha)}_{\boldsymbol{s}_i\oplus\mathbf{1}}\}^{\log 40T}_{\alpha=1}$ for $i = \mathsf{rec}+1, \ldots, \mathsf{rec}+n_o$ during the simulation, an adversary who flips $\geq t+1$ bits from $\{\boldsymbol{s}_i\}^{\mathsf{rec}+n_o}_{i=\mathsf{rec}+1}$ can only passes the check with probability $< 2^{-\kappa}$ using a similar analysis as in the proof of authenticity. Thus, the distributions of $\mathbf{Hyb}_{13}$ and $\mathbf{Hyb}_{12}$ are indistinguishable for $\mathcal{A}$.

$\mathbf{Hyb}_{14}$: In this hybrid, G does not compute his output by himself. Instead, he directly gets his outputs from $\mathcal{F}$, where the evaluator's inputs to $\mathcal{F}$ are reconstructed from the input sharings using error correction. Note that the set $\{\boldsymbol{s}_i, \boldsymbol{r}^{(1)}_{\boldsymbol{s}_i}, \ldots, \boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}_i}\}^{\mathsf{rec}+n_o}_{i=\mathsf{rec}+1}$ received from E match what G generates, we can regard that E honestly run the evaluation algorithm of the garbling scheme. From the correctness of the garbling scheme, G's outputs $\overline{y}_G$ of $\overline{C}$ obtained from the protocol execution is exactly $\mathsf{En}(y_G \oplus \mathsf{mask}, (\boldsymbol{r}^{(1)}_{\mathsf{en}}, \ldots, \boldsymbol{r}^{(W_O/k)}_{\mathsf{en}}))$ for G's output y_G from $\mathcal{F}$. The protocol output of G is decoded from $\mathsf{En}(y_G \oplus \mathsf{mask}, (\boldsymbol{r}^{(1)}_{\mathsf{en}}, \ldots, \boldsymbol{r}^{(W_O/k)}_{\mathsf{en}}))$ with De. Recall that

$$\mathsf{De}(\mathsf{En}(y_G \oplus \mathsf{mask}, (\boldsymbol{r}^{(1)}_{\mathsf{en}}, \ldots, \boldsymbol{r}^{(W_O/k)}_{\mathsf{en}}))) = y_G \oplus \mathsf{mask},$$

the protocol outputs $\mathsf{De}(\overline{y}_G) \oplus \mathsf{mask}$ is the same as the functionality output y_G. Thus, $\mathbf{Hyb}_{14}$ and $\mathbf{Hyb}_{13}$ have the same output distribution.

Note that $\mathbf{Hyb}_{14}$ is the ideal-world scenario, where the statistical difference between any two hybrids $\mathbf{Hyb}_i$ and $\mathbf{Hyb}_{i+1}$ $(i = 0, \ldots, 13)$ is no more than $\Theta(1)/2^{\kappa}$. Thus, the total difference between $\mathbf{Hyb}_0$ and

$\mathbf{Hyb}_{14}$ is still at most $\Theta(1)/2^\kappa$, so Π securely realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model against $\mathcal{A}$ when the evaluator E is corrupted.

J Cost Analysis for the 2PC

We analyze the communication cost of our 2PC as follows:

1. **Round 1:** In the first round, the parties only invoke the OT functionality. In particular, $O(W_I)$ instances of OT (with message length $O(\log T)$, same below) are required in Step 5, $O(\kappa/\log T)$ instances of OT are required in Step 6, $O(n^2(n+\kappa)/\log T) = \mathsf{poly}(\kappa, \log T)$ instances of OT are required in Step 7, and $n\ell_{\mathcal{O}}/\log T = \mathsf{poly}(\kappa, \log T)$ instances of OT are required in Step 9. In total, $O(W_I + \mathsf{poly}(\kappa, \log T))$ instances of OT with message length $O(\log T)$ are required.

2. **Round 2:** In the second round, the evaluator sends a $\Sigma_{\times\kappa'}$-sharing of the size $O(n\kappa') = \mathsf{poly}(\kappa, \log T)$, a set $\mathcal{U}$ that requires $n = \mathsf{poly}(\kappa, \log T)$ bits of communication, the wire labels for $O(n)$ servers' shares of $\Sigma_{\times\kappa'}$-sharing that has size $O(n \log T) = \mathsf{poly}(\kappa, \log T)$, and a $(\kappa + \log T)$-bit challenge s'. In total, the communication cost in the second round is $\mathsf{poly}(\kappa, \log T)$, which is independent of the circuit C.

3. **Round 3:** In the third round, the garbled circuit $\tilde{C}$ of size $O(|\overline{C}| \log T + \mathsf{depth}(\overline{C}) \cdot n^2 \log T) = O(|C| \log T + D(\kappa + \log T)^2 \log T)$ is sent (as $\mathsf{depth}(C) = O(D)$ and $|\overline{C}| = O(|C|)$). The garbler also sends VerSh, which contains a $\Sigma_{\times\kappa'}$-sharing, 2 $\Sigma^{(2)}_{\times\kappa'}$-sharings, and a $\Sigma^{(3)}_{\times\kappa'}$-sharing, where each of them has size of $O(n\kappa') = \mathsf{poly}(\kappa, \log T)$ bits. Moreover, the garbler needs to send $\mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}^{S_j}_{\mathsf{commit}}\}_{j\in\mathcal{V}}$, and $\{\mathsf{Sh}_{i,j}, \boldsymbol{r}^{(i,j)}_{\mathsf{comsh}}, \mathsf{Sh}_{j,i}, \boldsymbol{r}^{(j,i)}_{\mathsf{comsh}}\}_{i\in\mathcal{V}, j\in\{1,\ldots,n\}\setminus\{i\}}$. Sending $\mathcal{V}$ requires $O(n) = \mathsf{poly}(\kappa, \log T)$ bits. The size of $\{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$ and View^{S_j} for all $j \in \mathcal{V}$ is:

   - The total size of Δ^{S_j} for all $j \in \mathcal{V}$, which is $O(n \log T) = \mathsf{poly}(\kappa, \log T)$ bits.
   - The total size of $\lambda_w, k_{w,0}$ for all wires w excluding output wires of XOR/output gates in $\mathtt{Circ}^{S_j}$ for all $j \in \mathcal{V}$, together with all S_j's outputs from $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}^G_{\mathsf{input}}$. The size is no more than the label length $O(\log T)$ times the total size of $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}^G_{\mathsf{input}}$'s outputs and the number of AND gates in the servers' local computations, which is $O(\log T) \cdot O(|\overline{C}| + \mathsf{depth}(\overline{C}) \cdot n^2) = O(|C| \log T + D(\kappa + \log T)^2 \log T)$.
   - The sum of the size of each $S_j \in \mathcal{U}$'s shares of all $\{[\boldsymbol{r}^{(\alpha)}_{\boldsymbol{0}}]^{(3)}, [\boldsymbol{r}^{(\alpha)}_{\boldsymbol{1}} - \boldsymbol{r}^{(\alpha)}_{\boldsymbol{0}}]\}^{\log 40T}_{\alpha=1}$ corresponding to reconstructions of $\Sigma^{(2)}$-sharings whose receiver is not S_j. The total size is smaller than the total size of $\{[\boldsymbol{r}^{(\alpha)}_{\boldsymbol{0}}]^{(3)}, [\boldsymbol{r}^{(\alpha)}_{\boldsymbol{1}} - \boldsymbol{r}^{(\alpha)}_{\boldsymbol{0}}]\}^{\log 40T}_{\alpha=1}$ for all reconstructions of $\Sigma^{(2)}$-sharings in Π'_0, which is only a $O(\log T)$ times larger than the communication cost $O(|C| + D(\kappa + \log T)^2)$ of Π'_0.
   - The total size of all the sharings $\{[\boldsymbol{r}^{(\alpha)}_{\boldsymbol{0}}]^{(3)}, [\boldsymbol{r}^{(\alpha)}_{\boldsymbol{1}} - \boldsymbol{r}^{(\alpha)}_{\boldsymbol{0}}]\}^{\log 40T}_{\alpha=1}$ corresponding to reconstructions of $\Sigma^{(2)}$-sharings whose receiver is in $\mathcal{U}$. Similarly, the size is $O(|C| \log T + D(\kappa + \log T)^2 \log T)$ bits.
   - The total size of the shares of $[\boldsymbol{r}_1]_{\times\kappa'}$, $[\boldsymbol{r}_2]^{(2)}_{\times\kappa'}$, and $[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$ for servers in $\mathcal{U}$, which is $\mathsf{poly}(\kappa, \log T)$.

   For each $j \in \mathcal{V}$, the size of $\boldsymbol{r}^{S_j}_{\mathsf{commit}}$ and each $\boldsymbol{r}^{(i,j)}_{\mathsf{comsh}}$ is $O(\kappa + \log T)$. Thus, the total communication cost in this round is $O(|C| \log T + D(\kappa + \log T)^2 \log T + \mathsf{poly}(\kappa, \log T))$ bits.

4. **Round 4:** In the last round, only $\{\boldsymbol{s}_i, \boldsymbol{r}^{(1)}_{\boldsymbol{s}_i}, \ldots, \boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}_i}\}^{\mathsf{rec}+n_o}_{i=\mathsf{rec}+1}$ are sent from E to G. Note that $n_o = O(W_O/k) = O(|C|/n)$ (recall that W_O is the number of output wires of $\overline{C}$ to E, which is $O(|C|)$). Therefore, the total communication cost in this round is $O(|C|/n) \cdot O(n \cdot \log T) = O(|C| \log T)$.

In conclusion, the total communication complexity of our malicious 2PC protocol is $O(|C| \log T + D(\kappa + \log T)^2 \log T + \mathsf{poly}(\kappa, \log T))$ bits.

K Towards NISC

In this section, we provide our maliciously secure 2PC protocol in the following two settings (besides the assumption of a random oracle):

- In the preprocessing model of random bit-OT correlations against a computationally unbounded adversary with at most T random oracle calls (Section K.2);
- From a semi-malicious random bit-OT correlations can be replaced by a 2-round OT protocol with adaptive input choices in the common reference string (CRS) model (formally defined later) against a probabilistic polynomial-time adversary (Section K.4).

The construction requires only three one-way communication rounds, reducing the round complexity by one compared to the protocol in Section 6. Moreover, when only the evaluator receives output, the final round can be omitted entirely, yielding a non-interactive secure computation (NISC) protocol.

K.1 Brief Overview of the Construction

We first briefly overview how we additionally reduce the number of rounds to achieve a 3-round MPC from a 2-round OT protocol with a similar communication cost. In particular, the 2-round OT protocol can either be instantiated from either the random bit-OT preprocessing model (Section K.2) or directly using the assumed OT protocol in the CRS model (Section K.4).

The key idea is that during the parallel OT round of our 2PC protocol in Section 6, the evaluator serves as the receiver of the OTs. Therefore, when the parallel OTs are instantiated by the 2-round OT protocol, the evaluator will send the first-round OT messages. Note that the evaluator also serves as the message sender in the second round to send the authentication information of his OT inputs. Therefore, it's natural to let the evaluator send the authentication information together with the first-round OT messages. This is feasible since the first-round OT messages already commit to the evaluator's OT inputs (so that we do not need the evaluator's input labels to commit to the OT inputs), and the challenge s can be generated via Fiat-Shamir transformation [FS86].

Then, in the second round, the garbler will send the second-round OT messages together with the messages he sent in the third round of the protocol in Section 6. The only difference is that the challenge s' is generated via Fiat-Shamir [FS86] instead of the evaluator. This is essential since the garbled circuit is not committed in the first round this time. Finally, if the garbler has output from the circuit, the evaluator sends the (encoded) garbler's masked output in the last round to the garbler as in the protocol of Section 6.

K.2 Construction from Bit-OT Correlations

We first work on 2PC in the preprocessing model of random bit-OT correlations, which is the same as the model considered in the concurrent work [IJPS26] on NISC. We first formalize the random bit-OT correlations, which are modeled in the two-party functionality $\mathcal{F}_{\mathsf{bitROT}}$ shown below.

Functionality $\mathcal{F}_{\mathsf{bitROT}}$

The trusted party interacts with a sender S and a receiver R.

1. The trusted party randomly samples two messages $r_0, r_1 \in \{0,1\}$ and a bit $\beta \in \{0,1\}$. If R is corrupted, r_0, r_1 are chosen by S. If S is corrupted, β, r_β are chosen by R.
2. The trusted party sends (r_0, r_1) to S and sends (β, r_β) to R.

Figure 28: Functionality for random bit-OT correlation.

As observed in [GIS18], a 2-round semi-honest OT protocol (with message length ℓ_{OT}) can be constructed in the $\mathcal{F}_{\mathsf{bitROT}}$-hybrid model with information-theoretic security. Concretely speaking, S, R jointly hold a vector of ℓ_{OT} instances random bit-OTs, i.e. S holds $\boldsymbol{r}_0, \boldsymbol{r}_1 \in \{0,1\}^{\ell_{\mathsf{OT}}}$ and R holds $\boldsymbol{\beta}, \boldsymbol{r}_{\boldsymbol{\beta}} = \boldsymbol{r}_0 + (\boldsymbol{r}_1 - \boldsymbol{r}_0) * \boldsymbol{\beta}$.

The OT receiver R with OT input b sends $\boldsymbol{u} = \boldsymbol{b} \oplus \boldsymbol{\beta}$ (where $\boldsymbol{b} = (b, b, \ldots, b) \in \{0,1\}^{\ell_{\mathsf{OT}}}$) to the OT sender S as the first-round OT message, and then S with inputs $\boldsymbol{m}_0, \boldsymbol{m}_1$ sends $(\boldsymbol{x}_0, \boldsymbol{x}_1) = (\boldsymbol{m}_0 \oplus \boldsymbol{r}_{\boldsymbol{u}}, \boldsymbol{m}_1 \oplus \boldsymbol{r}_{\boldsymbol{u}\oplus\boldsymbol{1}})$ back as the second-round OT message. Then, the receiver R computes his output by $\boldsymbol{x}_b \oplus \boldsymbol{r}_{\boldsymbol{\beta}}$. We note that this OT protocol can be applied to our idea for reducing the number of rounds.

We now present the protocol construction formally. Let $\mathcal{O}$ be a random oracle with a sufficiently large (still $\mathsf{poly}(\kappa, \log T)$) output length. Similarly to Section 6, we consider the security with statistical error $\Theta(1)/2^{\kappa}$. By adding a sufficiently large constant on κ, the statistical error can then be reduced to $2^{-\kappa}$. The protocol Π runs Π_1 (Figure 29), Π_2 (Figure 30), and Π_3 (Figure 31) in order.

The First Round. The first-round protocol Π_1 is given below in Figure 29. In the first round, the two parties perform local computations to initialize the underlying MPC protocol. Then, the evaluator sends his first-round OT messages together with their authentication information.

Protocol Π_1

The protocol is run by a garbler G and an evaluator E.

1. **Circuit Transformation.** We use the same algorithms En, De as in Garble. For the original circuit $C(x_1, y_1) = (C_G(x_1, y_1), C_E(x_1, y_1))$, where $C_G(x_1, y_1)$ (resp. $C_E(x_1, y_1)$) is the output for G (resp. E), we use $\overline{C}$ to denote the circuit:
$$\begin{aligned}&\overline{C}(x_G \| \mathsf{mask} \| \boldsymbol{r}_{\mathsf{en}}^{(1)} \| \ldots \| \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}, x_E)\\ =&(\mathsf{En}(C_G(x_G, x_E) \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)})), C_E(x_G, x_E)).\end{aligned}$$
2. **Initializing.** Let Π_0' be the underlying protocol based on an $(n = 6t, t = 147 \log T + 172\kappa, k, \ell)$-LSSS Σ over $\mathbb{F}_2$ that runs among two clients G, E and n servers $S_1, \ldots, S_n$ to compute the circuit $\overline{C}$, where G's input is $x_G \| \mathsf{mask} \| \boldsymbol{r}_{\mathsf{en}}^{(1)} \| \ldots \| \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$, and E's input is x_E. Here x_G, x_E are G, E's inputs to C, respectively.
3. **Generating Preprocessing and Input Data.** E emulates $\mathcal{F}_{\mathsf{input}}^{E}$ to generate his input sharings in Π_0'.
4. **Preparing Random Bit-OT Correlations.** G serves as the sender and E serves as the receiver to invoke $\mathcal{F}_{\mathsf{bitROT}}$ $L \cdot (\log 40T + 1)$ times to prepare $L \cdot (\log 40T + 1)$ random bit-OT correlations, where L is the total size of E's input sharings.
5. **Generating Mask Sharing.** Let $\kappa' = n + \kappa$, E samples a random $\Sigma_{\times\kappa'}$-sharing as $[\boldsymbol{r}_0]_{\times\kappa'}$.
6. **Computing First-Round OT Messages.** For each bit b of E's inputs sharings, the parties consume $(\log 40T + 1)$ random bit-OT correlations, where G holds $\boldsymbol{r}_0, \boldsymbol{r}_1 \in \{0,1\}^{\log 40T}$ and E holds $\boldsymbol{\beta}, \boldsymbol{r}_{\boldsymbol{\beta}} = \boldsymbol{r}_0 + (\boldsymbol{r}_1 - \boldsymbol{r}_0) * \boldsymbol{\beta}$. Then, E computes a first round OT-message $\mathsf{ots}_1 = \boldsymbol{\beta} \oplus \boldsymbol{b}$ corresponding to this input bit b, where $\boldsymbol{b} = (b, b, \ldots, b) \in \{0,1\}^{\log 40T+1}$. Let the set of all the first-round OT messages ots_1 corresponding to the bits of server S_j's share of the input sharings (i.e. the inputs to $\mathtt{Circ}^{S_j}$) be $\mathsf{OT1Msg}_j$. E sends $\{\mathsf{OT1Msg}_j\}_{j=1}^n$ to G.
7. **Committing to OT Inputs.** For each $j = 1, \ldots, n$, let Input_j be the set containing:
   - inputs to $\mathtt{Circ}^{S_j}$ generated from $\mathcal{F}_{\mathsf{input}}^{E}$;
   - E's preprocessing data $\boldsymbol{\beta}, \boldsymbol{r}_{\boldsymbol{\beta}}$ corresponding to input bit b that is from S_j's shares of the input sharings.
   - S_j's share of $[\boldsymbol{r}_0]_{\times\kappa'}$.

   E samples a random string $\boldsymbol{r}_{\mathsf{comot}}^{S_j} \in \mathbb{F}_2^{\kappa+\log T}$ and queries the random oracle to get $\tau_j = \mathcal{O}(\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j)$. E then sends $\{\tau_j\}_{j=1}^n$ to G.
8. **Generating the Challenge.** Let the number of input Σ-sharings output by $\mathcal{F}_{\mathsf{input}}^{E}$ be k_0. E queries $\mathcal{O}$ with the challenge $s = \{\mathsf{OT1Msg}_j\}_{j=1}^n \| \{\tau_j\}_{j=1}^n$ and lets the first $\kappa' \cdot k_0$ bits of $\mathcal{O}(s)$ be $\theta_1, \ldots, \theta_{k_0} \in \mathbb{F}_{2^{\kappa'}}^{k_0}$.
9. **Computing Linear Combination.** E regards his input sharings $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ as $\Sigma_{\times\kappa'}$-sharings $[\boldsymbol{I}_1]_{\times\kappa'}, \ldots, [\boldsymbol{I}_{k_0}]_{\times\kappa'}$ (by embedding each bit into $\mathbb{F}_{2^{\kappa'}}$). Then, E computes:
$$[\boldsymbol{v}]_{\times\kappa'} = \sum_{i=1}^{k_0} \theta_i [\boldsymbol{I}_i]_{\times\kappa'} + [\boldsymbol{r}_0]_{\times\kappa'}$$

and sends it to G.

10. **Generating the Challenge for Opening Shares.** E queries $\mathcal{O}$ with $s\|[\boldsymbol{v}]_{\times\kappa'}$ and obtains a subset $\mathcal{U}$ of $t-\kappa$ servers among $S_1,\ldots,S_n$ from the first n bits of $\mathcal{O}(s\|[\boldsymbol{v}]_{\times\kappa'})$. E sends $\mathcal{U}$ to G.
11. **Opening Commitments for OT Inputs.** For each $S_j \in \mathcal{U}$, E sends Input_j and $\boldsymbol{r}_{\mathsf{comot}}^{S_j}$ to G.

Figure 29: The first round of our 2PC from random OT correlations.

The Second Round. The second-round protocol Π_2 is given in Figure 30. In this round, the garbler verifies the evaluator's OT inputs, responds to the OT messages, sends his garbled circuits, and sends the authentication information for the garbled circuits.

Protocol Π_2

1. **Generating Output Mask.** Let W_O be the number of output wires attached to G. At the beginning of the protocol, the garbler samples a random mask $\mathsf{mask} \in \mathbb{F}_2^{W_O}$ for his output. For each batch of k output wires indexed i attached to G, G samples a random n-bit string $\boldsymbol{r}_{\mathsf{en}}^{(i)}$.
2. **Generating Preprocessing and Input Data.** G emulates $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^{G}$ to generate the preprocessing data and his input sharings in Π_0'.
3. **Checking Evaluator's Input Sharings.** G queries $\mathcal{O}$ with $s = \{\mathsf{OT1Msg}_j\}_{j=1}^n \| \{\tau_j\}_{j=1}^n$ to get $\theta_1,\ldots,\theta_{k_0} \in \mathbb{F}_{2^{\kappa'}}$, and G also queries $\mathcal{O}$ with $s\|[\boldsymbol{v}]_{\times\kappa'}$. Then G verifies whether:
   - The set $\mathcal{U}$ received from E matches $\mathcal{O}(s\|[\boldsymbol{v}]_{\times\kappa'})$.
   - The first-round OT messages in $\{\mathsf{OT1Msg}_j\}_{S_j\in\mathcal{U}}$ are computed honestly from E's OT inputs and randomness from $\{\mathsf{Input}_j\}_{S_j\in\mathcal{U}}$.
   - $\tau_j = \mathcal{O}(\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j)$ holds for each $S_j \in \mathcal{U}$.
   - The sharing $[\boldsymbol{v}]_{\times\kappa'}$ is a valid $\Sigma_{\times\kappa'}$-sharing.
   - Corresponding to each bit b of each $S_j \in \mathcal{U}$'s share of E's input sharings, the random bit-OT preprocessing data $\boldsymbol{\beta}, \boldsymbol{r}_{\boldsymbol{\beta}}$ in $\{\mathsf{Input}_j\}_{S_j\in\mathcal{U}}$ and the received preprocessing data $\boldsymbol{r}_0, \boldsymbol{r}_1$ from $\mathcal{F}_{\mathsf{bitROT}}$ match bit-OT correlation defined by $\mathcal{F}_{\mathsf{bitROT}}$.

   If not, G aborts the protocol.
4. **Garbling the Circuit.** Assume that the output wires of $\overline{C}$ attached to G are batched into n_o batches, indexed $j = 1,\ldots,n_o$. For $i = 1,\ldots,\mathsf{rec}$, let the receiver of the i-th reconstruction of $\Sigma^{(2)}$-sharing during Π_0' online phase be R_i, and let the sharing to be reconstructed be $[\boldsymbol{s}_i]^{(2)}$. For each $j = 1,\ldots,n_o$, we let $[\boldsymbol{s}_{\mathsf{rec}+j}]^{(2)}$ to be the output sharing for the j-th batch of output wires attached to G, and we let the receiver $R_{\mathsf{rec}+j}$ to be the client G, as G holds the j-th batch of output wires. Then, G runs Steps 4(a)-4(c),5,6 of Garble and obtains the garbled circuit $\tilde{C}$ corresponding to Π_0'.
5. **Generating Auxiliary Sharings.** G generates a random $\Sigma_{\times\kappa'}$-sharing $[\boldsymbol{r}_1]_{\times\kappa'}$, a random $\Sigma^{(2)}_{\times\kappa'}$-sharing $[\boldsymbol{r}_2]^{(2)}_{\times\kappa'}$, a random $\Sigma^{(3)}_{\times\kappa'}$-sharing $[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$, and a random $\Sigma^{(2)}_{\times\kappa'}$-sharing $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ with an all-zero secret $\boldsymbol{o} = (0,0,\ldots,0) \in \mathbb{F}_{2^{\kappa'}}^k$.
6. **Computing Second-Round OT Messages.** For each bit b that serves as the i-th input to $\mathtt{Circ}^{S_j}$ generated by $\mathcal{F}_{\mathsf{input}}^{E}$, G parse the first-round OT message $\mathsf{ots}_1 = \boldsymbol{\beta} + \boldsymbol{b}$ (denoted by $\boldsymbol{u}$) from E's message sent in the first round and then computes $(X_{i,0}^{(j)} \oplus \boldsymbol{r}_{\boldsymbol{u}}, X_{i,1}^{(j)} \oplus \boldsymbol{r}_{\boldsymbol{u}\oplus\mathbf{1}})$ as the second-round OT message ots_2, where $\boldsymbol{r}_{\boldsymbol{u}}, \boldsymbol{r}_{\boldsymbol{u}\oplus\mathbf{1}}$ are computed from $\boldsymbol{r}_0, \boldsymbol{r}_1$ received from the same instances of $\mathcal{F}_{\mathsf{bitROT}}$ corresponding to b. Let the set of all the second-round OT messages ots_2 for the inputs to $\mathtt{Circ}^{S_j}$ be $\mathsf{OT2Msg}_j$. For each $j = 1,\ldots,n$, let Rand_j be the set containing G's preprocessing data $\boldsymbol{r}_0, \boldsymbol{r}_1$ for all instances of OTs prepared for these inputs to $\mathtt{Circ}^{S_j}$.
7. **Computing Servers' View.** For each $j \in \mathcal{V}$, let View^{S_j} be the set of:
   - Δ^{S_j};
   - $\lambda_w, k_{w,0}$ for all wires w excluding output wires of XOR/output gates in $\mathtt{Circ}^{S_j}$, all the outputs to S_j from $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^{G}$ excluding S_j's shares of the $\Sigma^{(2)}$-sharings whose secrets are held by other servers;

- S_j's shares of all $\{[\boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\mathbf{1}}^{(\alpha)} - \boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ corresponding to reconstructions of $\Sigma^{(2)}$-sharings whose receiver is not S_j;
- all the sharings $\{[\boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\mathbf{1}}^{(\alpha)} - \boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ corresponding to reconstructions of $\Sigma^{(2)}$-sharings whose receiver is S_j;
- S_j's shares of $[\boldsymbol{r}_1]_{\times\kappa'}$, $[\boldsymbol{r}_2]^{(2)}_{\times\kappa'}$, $[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$, and $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$.
- The set Rand_j.

Then, G samples a random string $\boldsymbol{r}_{\mathsf{commit}}^{S_j} \in \mathbb{F}_2^{\kappa+\log T}$ and queries the random oracle to get $c_j = \mathcal{O}(\mathsf{View}^{S_j} \| \boldsymbol{r}_{\mathsf{commit}}^{S_j} \| j)$. Moreover, for each pair of servers (S_i, S_j), let all the shares for S_j from $\mathcal{F}_{\mathsf{prep}}$ of the sharings whose secrets are held by S_i form a set $\mathsf{Sh}_{i,j}$. Then, G samples a random string $\boldsymbol{r}_{\mathsf{comsh}}^{(i,j)} \in \mathbb{F}_2^{\kappa+\log T}$ and queries the random oracle to get $\sigma_{i,j} = \mathcal{O}(\mathsf{Sh}_{i,j} \| \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)} \| i \| j)$.

8. **Computing Coefficients.** Let $\tilde{C} = ((F_1, L_1, d_1), \ldots, (F_n, L_n, d_n), F_{\mathsf{ct}})$, and let
$$\mathsf{GC} = (\tilde{C}, \{\mathsf{OT2Msg}_j\}_{j=1}^n, c_1, \ldots, c_n, \{\sigma_{i,j}, \sigma_{j,i}\}_{1 \le i < j \le n}).$$
Recall that $\kappa' = n + \kappa$. Let the number of Σ-sharings (resp. $\Sigma^{(2)}$-sharings, $\Sigma^{(3)}$-sharings, $\Sigma^{(2)}$-sharings with all-zero secrets) generated by G while emulating $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^G$ and during Step 4 of Garble be k_1 (resp. k_2, k_3, k_4). G queries $\mathcal{O}$ with the challenge GC and lets the first $\kappa'(k_1 + k_2 + k_3 + k_4)$ bits of $\mathcal{O}(\mathsf{GC})$ be $(\alpha_1, \ldots, \alpha_{k_1}, \beta_1, \ldots, \beta_{k_2}, \gamma_1, \ldots, \gamma_{k_3}, \eta_1, \ldots, \eta_{k_4})$ in $\mathbb{F}_{2^{\kappa'}}^{k_1+k_2+k_3+k_4}$.
9. **Generating Verification Sharings.** Let the k_1 Σ-sharings counted in the last step be $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, let the k_2 $\Sigma^{(2)}$-sharings be $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, let the k_3 $\Sigma^{(3)}$-sharings be $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, and let the k_4 $\Sigma^{(2)}$-sharings with all-zero secrets be $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$. G regards them as $\Sigma_{\times\kappa'}, \Sigma^{(2)}_{\times\kappa'}, \Sigma^{(3)}_{\times\kappa'}$-sharings (by embedding each bit into $\mathbb{F}_{2^{\kappa'}}$). Then, G computes VerSh as the set of
$$\Sigma_{i=1}^{k_1} \alpha_i [\boldsymbol{x}_i]_{\times\kappa'} + [\boldsymbol{r}_1]_{\times\kappa'}, \qquad \Sigma_{i=1}^{k_2} \beta_i [\boldsymbol{y}_i]^{(2)}_{\times\kappa'} + [\boldsymbol{r}_2]^{(2)}_{\times\kappa'},$$
$$\Sigma_{i=1}^{k_3} \gamma_i [\boldsymbol{z}_i]^{(3)}_{\times\kappa'} + [\boldsymbol{r}_3]^{(3)}_{\times\kappa'}, \text{ and } \Sigma_{i=1}^{k_4} \eta_i [\boldsymbol{o}_i]^{(2)}_{\times\kappa'} + [\boldsymbol{o}]^{(2)}_{\times\kappa'}.$$
10. **Generating the Challenge for Opening Views.** G queries $\mathcal{O}$ with $\mathsf{GC} \| \mathsf{VerSh}$ and lets the first n bits of the output denote a set $\mathcal{V}$ of $t/3$ indices in $\{1, \ldots, n\}$.
11. **Sending the Garbled Circuit.** G sends $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}\}_{j \in \mathcal{V}}, \{\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}, \mathsf{Sh}_{j,i}, \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}\}_{i \in \mathcal{V}, j \in \{1,\ldots,n\} \setminus \{i\}})$ to E.

Figure 30: The second round of our 2PC from random OT correlations.

The Third Round. The third-round protocol Π_3 is given in Figure 31. In this round, the evaluator verifies the authentication information of the garbled circuit and second-round OT messages. Finally, the evaluator obtains his own circuit outputs and sends the GC outputs to the garbler.

Protocol Π_3

1. **Verifying the Garbled Circuit.** Corresponding to each bit b of the evaluator's input sharing that serves as the i-th input to $\mathtt{Circ}^{S_j}$, the evaluator receives second round OT messages $(\boldsymbol{x}_0 = X_{i,0}^{(j)} \oplus \boldsymbol{r}_{\boldsymbol{u}}, \boldsymbol{x}_1 = X_{i,1}^{(j)} \oplus \boldsymbol{r}_{\boldsymbol{u} \oplus \mathbf{1}})$ and then computes the corresponding wire label by $\boldsymbol{x}_b \oplus \boldsymbol{r}_{\boldsymbol{\beta}}$. The evaluator E computes $(\alpha_1, \ldots, \alpha_{k_1}, \beta_1, \ldots, \beta_{k_2}, \gamma_1, \ldots, \gamma_{k_3}, \eta_1, \ldots, \eta_{k_4})$ from $\mathcal{O}(\mathsf{GC})$ checks whether the following hold:
   - The 4 sharings in VerSh are valid $\Sigma_{\times\kappa'}$, $\Sigma^{(2)}_{\times\kappa'}$, $\Sigma^{(3)}_{\times\kappa'}$-sharings and a $\Sigma^{(2)}_{\times\kappa'}$-sharing with an all-zero secret.
   - $c_j = \mathcal{O}(\mathsf{View}^{S_j} \| \boldsymbol{r}_{\mathsf{commit}}^{S_j} \| j)$ for each $j \in \mathcal{V}$, and $\sigma_{i,j} = \mathcal{O}(\mathsf{Sh}_{i,j} \| \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)} \| i \| j)$, $\sigma_{j,i} = \mathcal{O}(\mathsf{Sh}_{j,i} \| \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)} \| j \| i)$ for each $j \in \mathcal{V}, i \in \{1, \ldots, n\} \setminus \{j\}$.
   - For the output $\Sigma^{(2)}$-sharings from $\mathcal{F}_{\mathsf{prep}}$ to $j \in \mathcal{V}$ where S_j holds the secrets, the shares from $\{\mathsf{Sh}_{j,i}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$ form valid sharings, and the secret in View^{S_j} matches the sharings.
   - $\tilde{C}, \{\mathsf{OT2Msg}_j\}_{j \in \mathcal{V}}, d_{\mathsf{prep}}^E, \mathsf{VerSh}$ match $\{\mathsf{View}^{S_j}\}_{j \in \mathcal{V}}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i \in \mathcal{V}, j \in \{1,\ldots,n\} \setminus \{i\}}$. More concretely, the

following elements in $\tilde{C}, \{\mathsf{OT2Msg}_j\}_{j\in\mathcal{V}}, d^E_{\mathsf{prep}}, \mathsf{VerSh}$ can be computed from $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$:
  - (F_j, L_j, d_j) and the ciphertexts in F_{ct} generated in Step 6 of Garble for $R_i = S_j$.
  - The wire labels for S_j's shares of E's input sharings.
  - S_j's shares of the 4 sharings in VerSh.
  - The set $\mathsf{OT2Msg}_j$ of second-round OT messages.
- The obtained wire labels for the input shares of S_j matches the labels in View^{S_j}.
- $\mathcal{V}$ matches $\mathcal{O}(\mathsf{GC}\|\mathsf{VerSh})$.
- The preprocessing data in $\{\mathsf{Rand}_j\}_{j\in\mathcal{V}}$ and the received output from $\mathcal{F}_{\mathsf{bitROT}}$ form valid bit-OT correlation defined by $\mathcal{F}_{\mathsf{bitROT}}$.

2. **Evaluation.** E follows the evaluation algorithm Eval to evaluate the garbled circuit, except that E uses error correction to reconstruct the secrets (see Section H for the detailed evaluation process).
3. **Sending the Output.** After evaluating the garbled circuits, E reconstructs his own outputs from the output sharings via error correction and sends $\{\boldsymbol{s}_i, \boldsymbol{r}^{(1)}_{\boldsymbol{s}_i}, \ldots, \boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}_i}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ to G.
4. **Decoding the Output.** G runs the Decode to decode his output $\overline{y}_G$ of $\overline{C}$. Then, G checks whether $\overline{y}_G$ is a valid output of En. If not, G aborts the protocol. Otherwise, G computes his outputs of C by $\mathsf{De}(\overline{y}_G) \oplus \mathsf{mask}$.

Figure 31: The third round of our 2PC from random OT correlations.

We provide the result below.

Theorem 6 *Assuming $O(W_I \cdot \log T)$ random bit-OT correlations and a random oracle, there is a 3-round maliciously secure maliciously secure two-party computation protocol for any Boolean circuit C with communication of $O(|C| \log T + D(\log T + \kappa)^2 \log T + \mathsf{poly}(\kappa, \log T))$ bits against any computationally unbounded adversary with T random oracle queries, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the statistical security parameter. The protocol achieves statistical security with error $2^{-\kappa}$.*

When G has no output, Steps 3 and 4 in Π_3 are not required. Then, Π_3 only includes the evaluator's local computation, which gives a 2-round protocol, i.e. a non-interactive secure computation (NISC) protocol, with the same communication complexity.

Theorem 3 *Assuming $O(W_I \cdot \log T)$ random bit-OT correlations and a random oracle, there is a maliciously secure non-interactive secure computation (NISC) protocol for any Boolean circuit C with communication of $O(|C| \log T + D(\log T + \kappa)^2 \log T + \mathsf{poly}(\kappa, \log T))$ bits against any computationally unbounded adversary with T random oracle queries, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the statistical security parameter. The protocol achieves statistical security with error $2^{-\kappa}$.*

K.3 Security Proof for the Protocol from Random Bit-OT Correlations

We now prove the security of our 2PC protocol Π presented in Section K.2.

We prove the security by constructing an ideal adversary Sim. Then we will show that the output in the ideal world is indistinguishable from that in the real world against the computationally unbounded adversary $\mathcal{A}$ who has T RO queries using hybrid arguments. Note that $\mathcal{A}$ may corrupt either the garbler G or the evaluator E, so we prove the security under the two cases separately.

Security against a Corrupted Garbler. When G is corrupted, we construct the ideal adversary Sim below. We separate the description of Sim into the simulators $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ for the 3 rounds, respectively.

Simulator Sim_1

1. Sim initializes $\mathsf{ShCheck} = \mathsf{CompCheck} = 0$ and $\mathsf{Comp}_1 = \cdots = \mathsf{Comp}_n = 0$.
2. Sim follows the protocol to initialize the protocol Π_0'.
3. For each $j = 1, \ldots, n$, Sim samples τ_j randomly and sends it to G on behalf of E.
4. Sim samples a random $\Sigma_{\times\kappa'}$-sharing as $[\boldsymbol{v}]_{\times\kappa'}$ and samples a random subset of $t - \kappa$ servers as $\mathcal{U}$.
5. For each instance of OT for the inputs to $\mathtt{Circ}^{S_j}$ for each $S_j \in \mathcal{U}$, Sim faithfully emulates $\mathcal{F}_{\mathsf{bitROT}}$ to generate G's preprocessing data and randomly samples S_j's shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ (such that $\{\mathsf{Input}_j\}_{S_j \in \mathcal{U}}$ has been generated except S_j's share of $[\boldsymbol{r}_0]_{\times\kappa'}$). Then, Sim follows the protocol compute $\mathsf{OT1Msg}_j$. Corresponding to each input bit to $\mathtt{Circ}^{S_j}$ for each $S_j \notin \mathcal{U}$ generated from $\mathcal{F}_{\mathsf{input}}^E$, Sim samples a random $(\log 40T + 1)$-bit string as the first-round OT message $\boldsymbol{\beta} \oplus \boldsymbol{b}$. Then, Sim obtains $\mathsf{OT1Msg}_j$. Finally, Sim emulates $\mathcal{F}_{\mathsf{bitROT}}$ to send the G's preprocessing data to G and sends $\{\mathsf{OT1Msg}_j\}_{j=1}^n$ to G on behalf of E.
6. For each $j = 1, \ldots, n$, Sim samples the RO output τ_j randomly. Then, Sim sends $\{\tau_j\}_{j=1}^n$ to G on behalf of E. Whenever the first $\kappa + 2\log T$ bits of the sampled outputs of $\mathcal{O}$ for two different queries are the same, Sim aborts the simulation.
7. Sim follows the protocol to emulate the random oracle and compute the coefficients $\theta_1, \ldots, \theta_{k_0}$. Sim computes their shares of $[\boldsymbol{r}_0]_{\times\kappa'}$ by
$$[\boldsymbol{r}_0]_{\times\kappa'} = [\boldsymbol{v}]_{\times\kappa'} - \sum_{i=1}^{k_0} \theta_i [\boldsymbol{I}_i]_{\times\kappa'}.$$
8. Sim samples a random $(\kappa + \log T)$-bit string as $\boldsymbol{r}_{\mathsf{comot}}^{S_j}$ for each $S_j \in \mathcal{U}$. Sim then sends $\{\mathsf{Input}_j, \boldsymbol{r}_{\mathsf{comot}}^{S_j}\}_{S_j \in \mathcal{U}}$ to G on behalf of E. For each $S_j \in \mathcal{U}$, Sim sets the RO output for the query $\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j$ to τ_j.

Figure 32: The simulator for the first round when G is corrupted.

Simulator Sim_2

1. For each $j = 1, \ldots, n$, Sim follows the protocol to compute (F_j, L_j, d_j) and the ciphertexts in F_{ct} generated in Step 6 of Garble for $R_i = S_j$ from $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$ received by $\mathcal{O}$.
2. Sim emulates $\mathcal{O}$ to receive $\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}$ for each $j = 1, \ldots, n$ and $\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}$ for each pair of servers (S_i, S_j). Sim honestly emulates $\mathcal{O}$ to send the outputs to G. Then, Sim retrieve the corresponding committed $(\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j})$ for each $j = 1, \ldots, n$ and $(\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)})$ for each pair of servers (S_i, S_j) from the RO queries that match the commitments. If such committed inputs and randomness are not found from the RO queries for some commitment, Sim sets the committed values corresponding to this commitment to $\perp$.
3. Sim honestly emulates $\mathcal{O}$ to receive GC from G and outputs s' to G. Similarly, Sim honestly emulates $\mathcal{O}$ to receive $\mathsf{GC} \| \mathsf{VerSh}$ from G and outputs $\mathcal{V}$ to G.
4. Sim receives $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}\}_{j \in \mathcal{V}}, \{\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}, \mathsf{Sh}_{j,i}, \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}\}_{i \in \mathcal{V}, j \in \{1,\ldots,n\} \setminus \{i\}})$ from G on behalf of E. Then for each $j = 1, \ldots, n$, Sim checks whether:
   - (F_j, L_j, d_j) and the ciphertexts in F_{ct} for $R_i = S_j$ match what he computed in the last step.
   - The wire labels for S_j's shares of E's input sharings, the set Rand_j in View^{S_j}, and the first-round OT messages and preprocessing data Sim sent to G match the second-round OT messages in $\mathsf{OT2Msg}_j$.
   - $\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$ received in this step match what he received while emulating $\mathcal{O}$ in last two steps if $j \in \mathcal{V}$.
   - For the output $\Sigma^{(2)}$-sharings from $\mathcal{F}_{\mathsf{prep}}$ to S_j where S_j holds the secrets, the shares from $\{\mathsf{Sh}_{j,i}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$ form valid sharings, and the secret in View^{S_j} matches the sharings.
   - S_j's shares of the four sharings in VerSh are computed honestly from the shares in $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$.

   If not, Sim sets $\mathsf{Comp}_j = 1$.

5. If for more than $t/2$ indices $j \in \{1, \ldots, n\}$, it satisfies $\mathsf{Comp}_j = 1$, then Sim sets $\mathsf{CompCheck} = 1$.
6. If $\mathsf{CompCheck} = 0$, for each j with $\mathsf{Comp}_j = 0$, Sim obtains S_j's shares of $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, and $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$ from what G sends to $\mathcal{O}$ in the first round. Sim then checks whether these servers' shares of these sharings are all from valid $\Sigma/\Sigma^{(2)}/\Sigma^{(3)}$-sharings and $\Sigma^{(2)}$-sharings with all-zero secrets. If not, Sim sets $\mathsf{ShCheck} = 1$.

Figure 33: The simulator for the second round when G is corrupted.

Simulator Sim_3

1. Sim follows the protocol to verify the garbled circuit, excluding the check on input labels for E's input sharings. If the check does not pass, Sim aborts the protocol on behalf of E.
2. For the verification on input labels:
   - If for more than κ OT instances for input shares of P_j among all $j \in \mathcal{V}$, the wire labels for S_j's shares of E's input sharings, the preprocessing data from Rand_j in View^{S_j}, the first-round OT messages, and preprocessing data Sim sent to G do not match the second-round OT messages in $\mathsf{OT2Msg}_j$, Sim aborts the protocol on behalf of E.
   - If only for $\leq \kappa$ OT instances for input shares of P_j among all $j \in \mathcal{V}$, the wire labels for S_j's shares of E's input sharings, the preprocessing data from Rand_j in View^{S_j}, the first-round OT messages, and preprocessing data Sim sent to G do not match the second-round OT messages in $\mathsf{OT2Msg}_j$, Sim faithfully emulates $\mathcal{F}_{\mathsf{bitROT}}$ to sample E's preprocessing data among these ROT instances and then honestly emulates E to perform the check.
3. If $\mathsf{ShCheck} = 1$ or $\mathsf{CompCheck} = 1$, Sim aborts the simulation.
4. Sim reconstructs $\mathsf{mask}, \boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$ from the outputs of $\mathcal{F}_{\mathsf{input}}^{G}$ (received by $\mathcal{O}$) using error correction. Then, Sim uses then and the outputs of C for G to compute G's outputs of $\overline{C}$, i.e. $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$. Sim then computes the corresponding $\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)} = \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)} + (\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}) * \boldsymbol{s}_i$ for each $\alpha = 1, \ldots, \log 40T$, where $\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}, \boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)}$ are reconstructed from the sharings $[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]$ received by $\mathcal{O}$. Then, Sim sends $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ to G on behalf of E.
5. Sim outputs what $\mathcal{A}$ outputs.

Figure 34: The simulator for the third round when G is corrupted.

We construct the following hybrids:

$\mathbf{Hyb}_0$: In this hybrid, Sim gets E's inputs and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, Sim additionally sets $\mathsf{ShCheck} = \mathsf{CompCheck} = 0$ and $\mathsf{Comp}_1 = \cdots = \mathsf{Comp}_n = 0$ at the beginning of the simulation. This does not affect the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, while emulating the random oracle $\mathcal{O}$, Sim aborts the simulation if the first $\kappa + 2\log T$ bits of the RO outputs are the same for two different queries. Then, for each commitment generated by $\mathcal{O}$ from G, Sim can extract the committed inputs and randomness from the queries made by G. If such a query that matches the commitment is not found, Sim sets the committed inputs and randomness to $\perp$. Note that the RO output for each query is sampled randomly, so the probability that the first $\kappa + 2\log T$ bits of two outputs are the same is $1/2^{-(\kappa+2\log T)}$. Taking the union bound of all $< T^2$ possible pairs of queries, the probability is still less than $2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are computationally indistinguishable.

$\mathbf{Hyb}_3$: In this hybrid, Sim does not follow the protocol to sample $[\boldsymbol{r}_0]_{\times\kappa'}$ and compute $[\boldsymbol{v}]_{\times\kappa'}$. Instead, Sim samples $[\boldsymbol{v}]_{\times\kappa'}$ first and then computes $[\boldsymbol{r}_0]_{\times\kappa'}$ by

$$[\boldsymbol{r}_0]_{\times\kappa'} = [\boldsymbol{v}]_{\times\kappa'} - \sum_{i=1}^{k_0} \theta_i [\boldsymbol{I}_i]_{\times\kappa'}.$$

Since $[\boldsymbol{r}_0]_{\times\kappa'}$ is a uniformly random $\Sigma_{\times\kappa'}$-sharing, so is $[\boldsymbol{v}]_{\times\kappa'}$. Therefore, we only change the order of them without changing their distribution. Thus, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have the same output distribution.

$\mathbf{Hyb}_4$: In this hybrid, Sim does not follow the protocol to emulate the random oracle and compute $\{\tau_j\}_{j=1}^n$. Instead, Sim directly samples them randomly and then sets the RO outputs for the corresponding inputs $\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j$.

Note that when $\mathcal{A}$ does not query the random oracle with $\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j$, we do not change anything. Thus, the distribution only changes when $\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j$ is queried before the first-round message is sent. Since $\boldsymbol{r}_{\mathsf{comot}}^{S_j} \in \{0,1\}^{\kappa+\log T}$, each query made by $\mathcal{A}$ matches $\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j$ with a probability $\leq 2^{-(\kappa+\log T)}$. Taking the union bound of at most T queries, the probability is still $\leq 2^{-\kappa}$. Thus, the output distributions of $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ are computationally indistinguishable

$\mathbf{Hyb}_5$: In this hybrid, Sim does not follow the protocol to emulate the random oracle and compute $\mathcal{U}$. Instead, Sim directly samples it randomly and then sets the RO outputs for the corresponding input $\mathcal{O}(s \| [\boldsymbol{v}]_{\times\kappa'})$.

Note that when $\mathcal{A}$ does not query the random oracle with $s \| [\boldsymbol{v}]_{\times\kappa'}$, we do not change anything. Thus, the distribution only changes when $s \| [\boldsymbol{v}]_{\times\kappa'}$ is queried before the first-round message is sent. Since $s \| [\boldsymbol{v}]_{\times\kappa'}$ contains $\{\tau_j\}_{j=1}^n$, where each τ_j is a random string of length $> \kappa + \log T$, for the same reason as in $\mathbf{Hyb}_4$, the output distributions of $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ are computationally indistinguishable.

$\mathbf{Hyb}_6$: In this hybrid, while generating $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$, Sim first generates the shares for servers in $\mathcal{U}$ and then samples the other servers' shares based on these shares and the secrets. Note that $|\mathcal{U}| < t$, the shares for servers in $\mathcal{U}$ are independent of the secrets. Therefore, we only change the order of the generations for different servers' shares without changing their distributions. Thus, $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ have the same output distribution.

$\mathbf{Hyb}_7$: In this hybrid, after receiving $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}\}_{j\in\mathcal{V}})$ from G on behalf of E, for each $j = 1, \ldots, n$, Sim additionally checks whether:

- (F_j, L_j, d_j) and the ciphertexts in F_{ct} for $R_i = S_j$ match what he computed in the last step.
- The wire labels for S_j's shares of E's input sharings, the preprocessing data in the set Rand_j from View^{S_j}, and the first-round OT messages and preprocessing data Sim sent to G match the second-round OT messages in $\mathsf{OT2Msg}_j$.
- $\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$ received in this step match what he received while emulating $\mathcal{O}$ in last two steps if $j \in \mathcal{V}$.
- For the output $\Sigma^{(2)}$-sharings from $\mathcal{F}_{\mathsf{prep}}$ to S_j where S_j holds the secrets, the shares from $\{\mathsf{Sh}_{j,i}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$ form valid sharings, and the secret in View^{S_j} matches the sharings.
- S_j's shares of the four sharings in VerSh are computed honestly from the shares in $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$.

If not, Sim sets $\mathsf{Comp}_j = 1$. If for more than $t/2$ indices $j \in \{1, \ldots, n\}$, it satisfies $\mathsf{Comp}_j = 1$, then Sim sets $\mathsf{CompCheck} = 1$. This does not affect the output distribution. Thus, $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have the same output distribution.

$\mathbf{Hyb}_8$: In this hybrid, after the check on the servers' computation, if $\mathsf{CompCheck} = 0$, for each j with $\mathsf{Comp}_j = 0$, Sim additionally computes S_j's shares of $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$ from what $\mathcal{O}$ receives. Sim then checks whether these servers' shares of these sharings are all from valid $\Sigma/\Sigma^{(2)}/\Sigma^{(3)}$-sharings and valid $\Sigma^{(2)}$-sharings with all-zero secrets. If not, Sim sets $\mathsf{ShCheck} = 1$. This does not affect the output distribution. Thus, $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ have the same output distribution.

$\mathbf{Hyb}_9$: In this hybrid, Sim does not follow the protocol to emulate $\mathcal{F}_{\mathsf{bitROT}}$ to generate E's preprocessing data and compute E's first-round OT messages corresponding to each bit b of S_j's input shares (from E's input sharings) for $S_j \notin \mathcal{U}$. Instead, Sim samples random $(\log 40T + 1)$-bit strings as the first-round OT messages $\boldsymbol{u}$. Sim delays the generation of each input bit b of them before the evaluation, and the corresponding preprocessing data $\boldsymbol{\beta}$ is computed via $\boldsymbol{u} \oplus \boldsymbol{b}$ after b is generated.

Since $\boldsymbol{\beta}$ is a uniformly random string from $\mathcal{F}_{\mathsf{bitROT}}$, the corresponding first-round OT message $\boldsymbol{u} = \boldsymbol{\beta} \oplus \boldsymbol{b}$ is also uniformly random. Therefore, first sampling $\boldsymbol{u}$ and then computing $\boldsymbol{\beta}$ based on $\boldsymbol{u}$ and $\boldsymbol{b}$ only changes the order of generating $\boldsymbol{u}$ and $\boldsymbol{\beta}$ without changing their distributions. Moreover, since $S_j \notin \mathcal{U}$, the preprocessing data $\boldsymbol{\beta}$ is not used before the evaluation, so delaying the generation of each bit b of S_j's input shares (from E's input sharings) for $S_j \notin \mathcal{U}$ does not affect the output distribution. Thus, $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ have the same output distribution.

Note that corresponding to each bit b considered above, the preprocessing data $\boldsymbol{\beta}$ is not used during the whole simulation process, so Sim no longer generates it in future hybrids.

$\mathbf{Hyb}_{10}$: In this hybrid, for the verification of input labels, if only for $\leq \kappa$ bits from input shares of P_j among all $j \in \mathcal{V}$, the wire labels for them, the set Rand_j in View^{S_j}, the first-round OT messages and preprocessing data Sim sent to G do not match the second-round OT messages in $\mathsf{OT2Msg}_j$, Sim changes the way of computing E's inputs to C. Sim first randomly generates the $\leq \kappa$ shares containing these bits based on the shares for servers in $\mathcal{U}$. Since the shares are from randomly sampled Σ-sharings, these bits are t-wise independent of the secrets. Thus, any $\leq \kappa$ shares of them can be generated without knowing the secrets based on the $t - \kappa$ servers' shares that have been generated in the first round. Therefore, we only change the order of generating different input shares without changing their distributions. Thus, $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ have the same output distribution.

$\mathbf{Hyb}_{11}$: In this hybrid, for the verification of input labels, if for more than κ bits of P_j's shares of E's input sharings among all $j \in \mathcal{V}$, the wire labels for S_j's shares of E's input sharings, the randomness set Rand_j in View^{S_j}, the first-round OT messages and preprocessing data Sim sent to G do not match the second-round OT messages in $\mathsf{OT2Msg}_j$, Sim does not follow the protocol to check them. Instead, Sim directly aborts the protocol on behalf of E in this case. This only changes the output distribution when the real check does not find the inconsistency of ROT instances in this case, which only happens when G incorrectly commits his random bit-OT messages but the bit-OT correlations remains valid. This only happens with at most $1/2$ probability. Therefore, the $> \kappa$ incorrectly committed ROTs will cause the abort with at least $1 - 2^{-\kappa}$ probability, so the distribution only changes with probability $\leq 2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ are computationally indistinguishable.

$\mathbf{Hyb}_{12}$: In this hybrid, if the verification of E passes with $\mathsf{ShCheck} = 1$, Sim aborts the simulation. This only changes the distribution when there exists a set of $\leq t/2$ servers where each server S_j in them satisfies $\mathsf{Comp}_j = 1$, such that the other parties' shares of a Σ or $\Sigma^{(2)}$ or $\Sigma^{(3)}$-sharing or a $\Sigma^{(2)}$-sharing with an all-zero secret committed by G in the first round are not from a valid sharing, but they pass the check for the sharings. Note that $\geq n - t/2$ servers' shares of the sharings in VerSh must be computed from the committed shares, as $\mathsf{Comp} = 0$ for $\geq n - t/2$ parties. Assume the $\geq n - t/2$ servers' shares of a Σ-sharing $[\boldsymbol{x}_i]$ are not from a valid Σ-sharing, there is only one possible $\alpha_i \in \mathbb{F}_{2^{\kappa'}}$ that can make the sharings in VerSh valid when other coefficients are fixed. Otherwise, if there is an $\alpha_i' \neq \alpha_i$ that makes the $\geq n - t/2$ servers' shares of the sharings in VerSh valid, then the $n - t/2$ servers' shares of $(\alpha_i - \alpha_i') \cdot [\boldsymbol{x}_i]$ must comes from a valid Σ-sharing, which conflicts with the fact that the $\geq n - t/2$ servers' shares of $[\boldsymbol{x}_i]$ are invalid. If these servers' shares of a $\Sigma^{(2)}$-sharing or a $\Sigma^{(3)}$-sharing or a $\Sigma^{(2)}$-sharing of an all-zero secret are invalid, the case is similar. Therefore, if all coefficients are uniformly randomly sampled after the servers' views are committed, the output distribution only changes with probability $\leq 2^{-\kappa'}$ when the set of $n - t/2$ parties is fixed. Taking the union bound of $< 2^n$ all such sets of servers, the probability is still $\leq 2^{-(\kappa' - n)} = 2^{-\kappa}$.

Recall that the coefficients are determined by $\mathcal{O}(\mathsf{GC})$, where GC contains all the commitments for the servers' views. When $\mathcal{A}$ never queries GC before the servers' views are committed, the coefficients are uniformly random to him, and the distribution only changes with probability $\leq 2^{-\kappa}$. On the other hand, the commitments are uniformly random RO outputs (whose length $> \kappa + \log T$) before $\mathcal{A}$ queries the random oracle to generate them. Therefore, the probability that $\mathcal{A}$ queries $\mathcal{O}$ with GC before receiving the commitments is still no more than $2^{-\kappa}$, so the output distribution changes with $\leq 2^{-\kappa}$ in both cases. Thus, the distributions of $\mathbf{Hyb}_{12}$ and $\mathbf{Hyb}_{11}$ are computationally indistinguishable.

$\mathbf{Hyb}_{13}$: In this hybrid, if the verification of E passes with $\mathsf{CompCheck} = 1$, Sim aborts the simulation. This only changes the distribution when for $> t/2$ servers in the multiparty garbling process, G does not compute the garbled circuit from the committed views of them, or the committed labels of their output of

$\mathcal{F}_{\mathsf{prep}}, \mathcal{F}^G_{\mathsf{input}}$ together with the preprocessing data do not match the second-round OT messages. For all the cases where G does not compute the garbling of a server S_j from the committed view View^{S_j}, E can detect it if $j \in \mathcal{V}$. When $< 12\kappa$ servers' committed labels and the corresponding second-round OT messages do not match what G receives from $\mathcal{F}_{\mathsf{bitROT}}$, the distribution only changes when G does not compute the garbled circuit from the committed views of $> t/2 - 12\kappa > 70(\kappa + \log T)$ servers.

Recall that the set $\mathcal{V}$ is determined by $\mathcal{O}(\mathsf{GC}\|\mathsf{VerSh})$. That is to say, $\mathcal{V}$ is not determined before GC and VerSh are determined. Fixing GC and VerSh, the probability that $\mathcal{V}$ does not include the $> 18(\kappa + \log T)$ indices is no more than

$$\frac{n - 70(\kappa + \log T)}{n} \cdot \frac{n - 70(\kappa + \log T) - 1}{n - 1} \cdot \cdots \cdot \frac{n - 70(\kappa + \log T) - t/3 + 1}{n - t/3 + 1}$$
$$< \left(\frac{n - 70(\kappa + \log T)}{n}\right)^{t/3} < \left(\frac{11}{12}\right)^{49(\kappa + \log T)} < 2^{-(\kappa + \log T)} = T^{-1} \cdot 2^{-\kappa}.$$

Note that $\mathcal{A}$ can query $\mathcal{O}$ for T times, where each time $\mathcal{A}$ can fix a pair of GC and VerSh and learn whether this pair can pass the check. Taking the union bound of them, the probability of $\mathcal{A}$ correctly obtaining a pair of GC and VerSh matching the condition that will make the output distribution change is no more than $2^{-\kappa}$. Otherwise, either the pair of GC and VerSh sent by G does not match the condition, or the query $\mathsf{GC}\|\mathsf{VerSh}$ is not made by the adversary $\mathcal{A}$. If the query is not made, $\mathcal{V}$ is random to $\mathcal{A}$, and the probability that the distribution changes is no more than $T^{-1} \cdot 2^{-\kappa}$. In conclusion, the output distribution only changes with probability of $\Theta(1)/2^{\kappa}$.

On the other hand, note that when $> \kappa$ servers' committed labels and the corresponding second-round OT messages do not match what G receives from $\mathcal{F}_{\mathsf{bitROT}}$ for S_j with $k \in \mathcal{V}$, Sim will directly abort the protocol on behalf of E. Consider X_j to be 1 when S_j's committed labels and the corresponding second-round OT messages do not match what G receives from $\mathcal{F}_{\mathsf{bitROT}}$ and $S_j \in \mathcal{V}$. When $e \geq 12\kappa$ pairs of committed labels do not match the second-round OT messages in total, if $\mathcal{V}$ is random, $X_1, \ldots, X_n$ are independent random variables. Let $X = X_1 + \cdots + X_e$, then the expectation of X is $\mu = e/3 \geq 4\kappa$, and Sim will not directly abort the protocol because of this reason only when $X < \kappa$. The probability is

$$P(X < \kappa) \leq P(1 - 3\mu/4) \leq e^{-3^2\kappa/(2\cdot 4)} < 2^{-\kappa}$$

by Chernoff bound. For the same reason, the output distribution only changes with probability of $2^{-\kappa}$.

Thus, the distributions of $\mathbf{Hyb}_{13}$ and $\mathbf{Hyb}_{12}$ are computationally indistinguishable.

$\mathbf{Hyb}_{14}$: In this hybrid, Sim does not follow the protocol to evaluate the garbled circuit on behalf of E. Instead, Sim only follows that protocol to evaluate the garbled subcircuits for the $\geq n - t$ servers S_j with $\mathsf{Comp}_j = 0$ whose shares of the $\Sigma, \Sigma^{(2)}, \Sigma^{(3)}$-sharings and the $\Sigma^{(2)}$-sharings with all-zero secrets are all from valid sharings. The secrets of $\Sigma^{(2)}, \Sigma^{(3)}$-sharings required to reconstruct in the evaluation process are reconstructed from the shares for these servers. Note that from the correctness of our garbling scheme, these servers' shares of the $\Sigma^{(2)}, \Sigma^{(3)}$-sharings reconstructed in the evaluation process are valid. The only difference from the original evaluation process is that Sim does not evaluate the sub-circuits for the other servers, and he does not use error correction to reconstruct the secrets for these sharings. Since the $\Sigma^{(2)}, \Sigma^{(3)}$-sharings have error-correction properties for up to t errors, no matter how the other t servers' garbled sub-circuits are evaluated, the secrets of these sharings are fixed, so we do not change any message sent to G. Thus, $\mathbf{Hyb}_{14}$ and $\mathbf{Hyb}_{13}$ have the same output distribution.

$\mathbf{Hyb}_{15}$: In this hybrid, Sim additionally computes the valid shares and garbled sub-circuits for the $\leq t$ servers, excluding the $\geq n - t$ servers taken in the last hybrid, from the committed views to them and simulates the whole evaluation process honestly using these garbled sub-circuits. Since the secrets of the $\Sigma^{(2)}, \Sigma^{(3)}$-sharings reconstructed in the evaluation process have already been determined by the other $n - t$ parties, we do not change any message sent to G. Thus, $\mathbf{Hyb}_{15}$ and $\mathbf{Hyb}_{14}$ have the same output distribution.

$\mathbf{Hyb}_{16}$: In this hybrid, Sim does not follow the protocol to evaluate the circuit and sends the garbled output $\{\boldsymbol{s}_i, \boldsymbol{r}^{(1)}_{\boldsymbol{s}_i}, \ldots, \boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}_i}\}_{R_i = G}$ to G on behalf of E. Instead, Sim extracts the output of $\mathcal{F}^G_{\mathsf{input}}$ from the commitments generated by G and reconstructs the inputs of G together with $\mathsf{mask}, \boldsymbol{r}^{(1)}_{\mathsf{en}}, \ldots, \boldsymbol{r}^{(W_O/k)}_{\mathsf{en}}$. Sim

then sends the inputs of G to $\mathcal{F}$ and receives G's output. Then, Sim uses $\mathsf{mask}, \boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$ and the circuit outputs of G to compute G's outputs of $\overline{C}$, i.e., $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$. Sim then computes the corresponding $\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)} = \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)} + (\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}) * \boldsymbol{s}_i$ for each $\alpha = 1, \ldots, \log 40T$, where $\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}, \boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)}$ are reconstructed from the sharings $[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]$ received by $\mathcal{O}$. In addition, E does not compute his output by himself. Instead, he directly gets his outputs from $\mathcal{F}$. Note that the shares of the input sharings generated by E for the $\geq n - t$ servers taken in $\mathbf{Hyb}_{16}$ and their corresponding wire labels are not used for the simulation. Sim does not generate them in this hybrid.

Since the garbling and evaluation in $\mathbf{Hyb}_{15}$ completely follow the protocol, from the correctness of the garbling scheme, the outputs of E computed in the two hybrids are the same. If G follows the protocol to compute his output, he will also get the correct output. Note that in the explicit protocol of Π_0, when $\mathsf{mask}, \boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$ are fixed, the computation from $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ to G's output is deterministic and invertible. Furthermore, when $\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}, \boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)}$ are fixed by the commitments, $\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}$ is also determined by $\boldsymbol{s}_i$. Therefore, we only change the order of generating G's output and the messages $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ sent to G without changing their distributions. Thus, $\mathbf{Hyb}_{16}$ and $\mathbf{Hyb}_{15}$ have the same output distribution.

Note that $\mathbf{Hyb}_{16}$ is the ideal-world scenario, where the statistical difference between any two hybrids $\mathbf{Hyb}_i$ and $\mathbf{Hyb}_{i+1}$ $(i = 0, \ldots, 15)$ is no more than $\Theta(1)/2^\kappa$. Thus, the total difference between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_{16}$ is still at most $\Theta(1)/2^\kappa$, so Π securely realizes $\mathcal{F}$ with computational security when the garbler G is corrupted.

Security against a Corrupted Evaluator. When E is corrupted, we construct the ideal adversary Sim below. We separate the description of Sim into the simulators $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ for the 3 rounds, respectively.

Simulator Sim_1

1. Sim faithfully emulates $\mathcal{F}_{\mathsf{bitROT}}$ to receive the output of E for each invocation of $\mathcal{F}_{\mathsf{bitROT}}$ from $\mathcal{A}$ and send the preprocessing data to E.
2. Sim follows the protocol to do the initialization and get the protocol Π_0'. Then, Sim sets $\mathcal{C} = \emptyset$, $\mathsf{ShCheck} = 0$, $\mathsf{CompCheck} = 0$, and $\mathsf{Comp}_j = \mathsf{Corr}_j = 0$ for each $j = 1, \ldots, n$.
3. Sim runs Steps 3-4 and Steps 6-8 of $\mathsf{Sim}_{\mathsf{pri}}$ (similar to $\mathsf{Sim}_{\mathsf{obl}}$) and obtains the garbled circuit $\tilde{C}$. In particular, Sim does not generate d_{mac}. Sim still honestly emulates $\mathcal{O}$.
4. From this step, once $\mathcal{A}$ queries a random oracle with query q:
   - If $q \notin Q_j$ for all $j \in \{1, \ldots, n\}$, Sim does nothing.
   - If q is in some Q_j, then Sim sets $\mathsf{Corr}_j = 1$ and adds S_j into $\mathcal{C}$. If $|\mathcal{C}| > 2t/3$ with $\mathsf{CompCheck} = \mathsf{ShCheck} = 0$, Sim aborts the simulation.

   Whenever Sim samples a random oracle output, if the first $\kappa + 2\log T$ bits of the random oracle outputs for two different queries, Sim aborts the simulation.
5. Sim receives $\{\tau_j\}_{j=1}^n$ from E and then extracts $\{\mathsf{Input}_j, \boldsymbol{r}_{\mathsf{comot}}^{S_j}\}_{j=1}^n$ from the RO queries it received. If such committed inputs and randomness are not found from the RO queries for some commitment, Sim sets the committed values corresponding to this commitment to $\perp$.
6. Sim receives $[\boldsymbol{v}]_{\times\kappa'}, \mathcal{U}$, and $\{\mathsf{Input}_j, \boldsymbol{r}_{\mathsf{comot}}^{S_j}\}_{S_j \in \mathcal{U}}$ from E.
7. Sim checks whether each S_j's share of $[\boldsymbol{v}]_{\times\kappa'}$ is computed honestly from S_j's shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$ (extracted from E's commitments). Sim also checks whether the first-round OT messages corresponding to S_j's input shares are computed honestly from the committed shares and the preprocessing data sent to E. If not, Sim sets $\mathsf{Comp}_j = 1$ and includes S_j into $\mathcal{C}$. Then:
   - If $|\mathcal{C}| \leq 2t/3$, then Sim runs Sim_0 to compute S_j's view based on the views other parties in $\mathcal{C}$. For each $i = 1, \ldots, \mathsf{rec} + n_o$, Sim samples S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for the i-th reconstruction based on the shares for servers in $\mathcal{C}$ and computes S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ accordingly by:

$$[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)} = [\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)} \oplus [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}] \otimes [\boldsymbol{s}_i]^{(2)}.$$

– If $|\mathcal{C}| > 2t/3$ and there are only $\leq t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim aborts the simulation.

If there are $> t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim sets $\mathsf{CompCheck} = 1$.

8. If $\mathsf{CompCheck} = 0$, for all j with $\mathsf{Comp}_j = 0$, Sim checks whether these S_j' shares of the sharings $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ are all from valid Σ-sharings.
   – If not, Sim sets $\mathsf{ShCheck} = 1$.
   – Otherwise, Sim reconstructs the secrets, i.e., E's inputs to C, from these sharings. Sim then sends E's inputs to $\mathcal{F}$ and learns E's outputs of C, which is equal to E's output of $\overline{C}$.

Figure 35: The simulator for the first round when E is corrupted.

Simulator Sim_2

1. Sim follows the protocol to perform the check on the evaluator's input sharings. If the check fails, Sim aborts the protocol on behalf of G.
2. If $\mathsf{ShCheck} = 1$ or $\mathsf{CompCheck} = 1$ but the check passes, Sim aborts the simulation.
3. Sim generates n random strings as $c_1, \ldots, c_n$ and $n(n-1)$ random sharings as $\sigma_{i,j}$ for all pairs of servers (S_i, S_j), where the length of the strings matches the output length of $\mathcal{O}$.
4. Sim generates a random $\Sigma_{\times\kappa'}$-sharing, a random $\Sigma^{(2)}_{\times\kappa'}$-sharing, and a random $\Sigma^{(3)}_{\times\kappa'}$-sharing as the three sharings in VerSh.
5. For each $S_j \in \mathcal{C}$, Sim samples all the wire masks for $\mathtt{Circ}^{S_j}$ excluding the wire masks for output wires of XOR/output gates. Sim also samples the shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for parties in $\mathcal{C}$. Then, Sim honestly emulates $\mathcal{O}$ to compute all the outputs for queries in Q_j for all $S_j \in \mathcal{C}$, and then Sim updates the ciphertexts in $\{F_j\}_{S_j \in \mathcal{C}}$ and F_{ct} using these query outputs. In particular, for each ciphertext of the form $\mathcal{O}(q) \oplus \mathsf{m}$ for some $q \in Q_j$ with $S_j \in \mathcal{C}$, m can be obtained by honestly running the garbling process for S_j, and then the ciphertext can be computed from $\mathcal{O}(q)$ and m. Now Sim has already got GC and Versh.
6. Sim honestly samples the set $\mathcal{V}$. For each $j \in \mathcal{V}$, Sim sets $\mathsf{Corr}_j = 1$ and includes S_j into $\mathcal{C}$. Then:
   – If $|\mathcal{C}| \leq t$, then Sim runs Sim'_0 to compute S_j's view based on the views other parties in $\mathcal{C}$. In particular, E is set to be corrupted. E's output from of $\overline{C}$ is set to be his output from $\mathcal{F}$. While emulating $\mathcal{F}^E_{\mathsf{input}}$, the outputs are set to be the input sharings extracted from the first bit of $\boldsymbol{\beta} \oplus \boldsymbol{b}$ and $\boldsymbol{\beta}$ after error correction. For each $i = 1, \ldots, \mathsf{rec} + n_o$, Sim samples S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for the i-th reconstruction based on the shares for servers in $\mathcal{C}$ and computes S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ accordingly by:
$$[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)} = [\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)} \oplus [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}] \otimes [\boldsymbol{s}_i]^{(2)}.$$
   – If $|\mathcal{C}| > t$, Sim aborts the simulation.

   From this step, Sim follows Step 9 of $\mathsf{Sim}_{\mathsf{pri}}$ to emulate $\mathcal{O}$. More concretely, once $\mathcal{A}$ queries a random oracle with query q:
   – If $q \notin Q_j$ for all $j \in \{1, \ldots, n\}$, $\mathsf{Sim}_{\mathsf{pri}}$ honestly emulates the random oracle.
   – If q is in some Q_j, then if $S_j \notin \mathcal{C}$, $\mathsf{Sim}_{\mathsf{pri}}$ sets $\mathsf{Corr}_j = 1$ and adds S_j into $\mathcal{C}$. Then:
     * If $|\mathcal{C}| > t$, $\mathsf{Sim}_{\mathsf{pri}}$ aborts the simulation.
     * Otherwise, $\mathsf{Sim}_{\mathsf{pri}}$ runs Sim'_0 to compute S_j's view suring Π'_0 based on the views other parties in $\mathcal{C}$. Then, for each $i = 1, \ldots, \mathsf{rec} + n_o$, $\mathsf{Sim}_{\mathsf{pri}}$ samples S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for the i-th reconstruction based on the shares for servers in $\mathcal{C}$ and computes S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ accordingly by:
$$[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)} = [\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)} \oplus [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}] \otimes [\boldsymbol{s}_i]^{(2)}.$$

   Then the response is determined via the following process:
   * If the queried oracle is $\mathcal{O}_1$, the response to q is determined by the equation for computing a ciphertext, which is of the form
$$\mathsf{ct} = \mathcal{O}_1(q) \oplus \mathsf{m}$$

for a message m that can be computed from S_j's view and the sampled wire labels. Note that all the ciphertexts have already been sampled. Then, the response can be computed by $\mathsf{ct} \oplus m$. The remaining bits of the output from $\mathcal{O}$ for the same query are then randomly sampled. Then the output of $\mathcal{O}$ for this query is fixed.

* If the queried oracle is $\mathcal{O}_2$, the response to q is also determined by the equation for computing a ciphertext, which is of the form

$$\mathsf{ct} = \mathcal{O}_2(q) \oplus \mathsf{m}$$

for a message m that can be computed from S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$. Note that all the ciphertexts have already been sampled. Then, the response can also be computed by $\mathsf{ct} \oplus m$. The remaining bits of the output from $\mathcal{O}$ for the same query are then randomly sampled. Then the output of $\mathcal{O}$ for this query is fixed.

7. For each server $S_j \in \mathcal{C}$, Sim samples G's outputs of $\mathcal{F}_{\mathsf{bitROT}}$ based on the outputs of $\mathcal{F}_{\mathsf{bitROT}}$ to E. Then, Sim follows the protocol to compute these second-round OT messages. For each server $S_j \notin \mathcal{C}$, Sim extracts b from $\boldsymbol{\beta} \oplus \boldsymbol{b}$ and $\boldsymbol{\beta}$ and computes $\boldsymbol{x}_b = k_{w,b\oplus\lambda_w} \| (b \oplus \lambda_w)$ for each bit b of S_j's share from E's input sharings. Then, Sim samples $\boldsymbol{x}_{b\oplus 1}$ randomly and obtains the second-round OT message ots_2 corresponding to b. Sim sends it to E on behalf of G. In addition, Sim sets the RO output of $\mathcal{O}(\mathsf{GC}\|\mathsf{Versh})$ to $\mathcal{V}\|\mathsf{randbits}$, where $\mathsf{randbits}$ is a randomly sampled string that matches the output length of $\mathcal{O}$. If $\mathsf{GC}\|\mathsf{Versh}$ has been queried by $\mathcal{A}$, Sim aborts the simulation.
8. For each $S_j \in \mathcal{C}$, Sim computes S_j's shares of $[\boldsymbol{r}_1]_{\times\kappa'}, [\boldsymbol{r}_2]^{(2)}_{\times\kappa'}, [\boldsymbol{r}_3]^{(3)}_{\times\kappa'}, [\boldsymbol{o}]^{(2)}_{\times\kappa'}$ from:
   - The four sharings in VerSh;
   - The coefficients $\{\alpha_i\}_{i=1}^{k_1}, \{\beta_i\}_{i=1}^{k_2}, \{\gamma_i\}_{i=1}^{k_3}, \{\eta_i\}_{i=1}^{k_4}$ obtained from $\mathcal{O}(\mathsf{GC})$;
   - S_j's shares of $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, and $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$ obtained from the view of S_j during the execution of Π'_0 and S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}], [\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ that has been generated.

   Now, Sim has obtained $\{\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}\}_{j\in\mathcal{V}}$.
9. For each $j \in \mathcal{V}$, Sim samples a random $(\kappa + \log T)$-bit string as $\boldsymbol{r}_{\mathsf{commit}}^{S_j}$. If $\mathsf{View}^{S_j}\|\boldsymbol{r}_{\mathsf{commit}}^{S_j}\|j$ has been queried by $\mathcal{A}$, then Sim aborts the simulation. Otherwise, Sim sets the output of $\mathcal{O}(\mathsf{View}^{S_j}\|\boldsymbol{r}_{\mathsf{commit}}^{S_j}\|j)$ to c_j. Similarly, for each $j \in \mathcal{V}$ and $i > j$ or $i \notin \mathcal{V}$, Sim samples 2 random $(\kappa + \log T)$-bit strings as $\boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}, \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}$. If $\mathsf{Sh}_{i,j}\|\boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}\|i\|j$ or $\mathsf{Sh}_{j,i}\|\boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}\|j\|i$ has been queried by $\mathcal{A}$, then Sim aborts the simulation. Otherwise, Sim sets the output of $\mathcal{O}(\mathsf{Sh}_{i,j}\|\boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}\|i\|j)$ and $\mathsf{Sh}_{j,i}\|\boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}\|j\|i$ to $\sigma_{i,j}$ and $\sigma_{j,i}$ respectively. Finally, Sim sends $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}\}_{j\in\mathcal{V}}, \{\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}, \mathsf{Sh}_{j,i}, \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}\}_{i\in\mathcal{V}, j\in\{1,\ldots,n\}\setminus\{i\}})$ to E on behalf of G.

Figure 36: The simulator for the second round when E is corrupted.

Simulator Sim_3

1. Sim receives $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ from E on behalf of G.
2. Sim checks whether $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ matches what Sim generates. If not, Sim aborts the protocol on behalf of G.
3. Sim outputs what $\mathcal{A}$ outputs.

Figure 37: The simulator for the third round when E is corrupted.

We construct the following hybrids:

$\mathbf{Hyb}_0$: In this hybrid, Sim gets G's inputs and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, Sim additionally sets $\mathcal{C} = \emptyset$, $\mathsf{ShCheck} = 0$, $\mathsf{CompCheck} = 0$, and $\mathsf{Comp}_j = \mathsf{Corr}_j = 0$ for each $j = 1, \ldots, n$ at the beginning of the simulation. This does not affect the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, while emulating the random oracle $\mathcal{O}$, Sim aborts the simulation if the first

$\kappa + 2\log T$ bits of the RO outputs are the same for two different queries. Then, for each commitment generated by $\mathcal{O}$ from G, Sim can extract the committed inputs and randomness from the queries made by G. If such a query that matches the commitment is not found, Sim sets the committed inputs and randomness to $\perp$. Note that the RO output for each query is sampled randomly, so the probability that the first $\kappa+2\log T$ bits of two outputs are the same is $1/2^{-(\kappa+2\log T)}$. Taking the union bound of all $< T^2$ possible pairs of queries, the probability is still less than $2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are computationally indistinguishable.

$\mathbf{Hyb}_3$: In this hybrid, Sim follows $\mathsf{Sim}_{\mathsf{pri}}$ to generate the sets $Q_1, \ldots, Q_n$ at the beginning, once $\mathcal{A}$ queries a random oracle with query q:

- If $q \notin Q_j$ for all $j \in \{1, \ldots, n\}$, Sim does nothing.
- If q is in some Q_j, then Sim sets $\mathsf{Corr}_j = 1$ and adds S_j into $\mathcal{C}$. If $|\mathcal{C}| > 2t/3$ with $\mathsf{CompCheck} = \mathsf{ShCheck} = 0$, Sim aborts the simulation.

This only changes the output distribution when $\mathcal{A}$ queries a query in $> 2t/3 > \kappa$ sets among $Q_1, \ldots, Q_n$. As argued in the proof of privacy for our garbling scheme, this only happens with $< 2^{-\kappa}$ probability. Thus, the distributions of $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ are computationally indistinguishable.

$\mathbf{Hyb}_4$: In this hybrid, Sim does not follow the protocol to generate the wire labels for the sharings $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$. Instead, for each wire w whose wire value is from S_j's shares of these sharings, Sim first samples wire label $k_{w,\lambda_w \oplus b_w} \| (\lambda_w \oplus b_w) \in \mathbb{F}_2^{\log 40T+1}$ randomly, where b_w is the wire value extracted from the commitments received from E. Then, Sim computes the other label by $k_{w,\lambda_w \oplus b_w \oplus} = k_{w,\lambda_w \oplus b_w} \oplus \Delta^{S_j}$ and computes the wire masks by $\lambda_w = (\lambda_w \oplus b_w) \oplus b_w$. Since $k_{w,0}$ and λ_w are uniformly randomly distributed, so is $k_{w,\lambda_w \oplus b_w} \| (\lambda_w \oplus b_w)$. We only change the order of sampling:

- λ_w and $\lambda_w \oplus b_w$;
- $k_{w,0}$ and $k_{w,1}$ when $\lambda_w \oplus b_w = 1$

without changing their distributions. Thus, $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ have the same output distribution.

$\mathbf{Hyb}_5$: In this hybrid, after receiving $[\boldsymbol{v}]_{\times\kappa'}, \mathcal{U}$ from E, Sim additionally checks whether each S_j's share of $[\boldsymbol{v}]_{\times\kappa'}$ is computed honestly from S_j's shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$ (extracted from the commitments received from E). If not, Sim sets $\mathsf{Comp}_j = 1$ and includes S_j into $\mathcal{C}$. If there are $\geq t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim sets $\mathsf{CompCheck} = 1$. This doesn't affect the output distribution. Thus, $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have the same output distribution.

$\mathbf{Hyb}_6$: In this hybrid, after the check added in the last hybrid, if $|\mathcal{C}| > 2t/3$ and there are only $\leq t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim aborts the simulation. Note that a server S_j with $\mathsf{Comp}_j \neq 1$ is added into $\mathcal{C}$ only because a query in Q_j has been queried. Note that until this step, Sim hasn't used the queries in Q_j to generate any messages sent to E before one of them is queried if $\mathsf{Comp}_j = 0$, which is the same case as in $\mathbf{Hyb}_2$. For the same reason, this happens with $< 2^{-\kappa}$ probability. Thus, the distributions of $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ are computationally indistinguishable.

$\mathbf{Hyb}_7$: In this hybrid, after the check on the servers' computation, if $\mathsf{CompCheck} = 0$, for each j with $\mathsf{Comp}_j = 0$, Sim additionally checks whether these servers S_j's shares of these sharings (extracted from the commitments received from E) are all from valid Σ-sharings. If not, Sim sets $\mathsf{ShCheck} = 1$. This does not affect the output distribution. Thus, $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have the same output distribution.

$\mathbf{Hyb}_8$: In this hybrid, if the verification of E passes with $\mathsf{ShCheck} = 1$, Sim aborts the simulation. This only changes the distribution when there exists a set of $\leq t/3$ servers where each server S_j in them satisfies $\mathsf{Comp}_j = 1$, such that the other parties' shares of an input Σ-sharing committed by E in the first round are not from a valid sharing, but they pass the check for the sharings. Note that $\geq n - t/3$ servers' shares of the sharing $[\boldsymbol{v}]_{\times\kappa'}$ must be computed from the committed shares, as $\mathsf{Comp} = 0$ for $\geq n - t/3$ parties. Assume the $\geq n - t/3$ servers' shares of a Σ-sharing $[\boldsymbol{I}_i]$ are not from a valid Σ-sharing, there is only one possible $\alpha_i \in \mathbb{F}_{2^{\kappa'}}$ that can make the sharings in VerSh valid when other coefficients are fixed. Otherwise, if there is an $\alpha_i' \neq \alpha_i$ that makes the $\geq n - t/3$ servers' shares of the sharing $[\boldsymbol{v}]_{\times\kappa'}$ valid, then the $n - t/3$ servers' shares of $(\alpha_i - \alpha_i') \cdot [\boldsymbol{I}_i]$ must comes from a valid Σ-sharing, which conflicts with the fact that the

$\geq n - t/3$ servers' shares of $[\boldsymbol{I}_i]$ are invalid. Therefore, if all coefficients are uniformly randomly sampled after the servers' views are committed, the output distribution only changes with probability $\leq 2^{-\kappa'}$ when the set of $n - t/3$ parties is fixed. Taking the union bound of $< 2^n$ all such sets of servers, the probability is still $\leq 2^{-(\kappa'-n)} = 2^{-\kappa}$.

Recall that the coefficients are determined by $\mathcal{O}(s)$, where s is sampled randomly after the servers' views are committed unless $\mathcal{A}$ queries the random oracle with s before $\{\tau_j\}_{j=1}^n$ are sent to $\mathcal{A}$. When $\mathcal{A}$ never queries s before the servers' views are committed, the coefficients are uniformly random to him, and the distribution only changes with probability $\leq 2^{-\kappa}$. On the other hand, $\{\tau_j\}_{j=1}^n$ are randomly sampled RO outputs, and s contains them. Therefore, the probability that each query made by $\mathcal{A}$ to the random oracle before $\{\tau_j\}_{j=1}^n$ are sampled and sent to $\mathcal{A}$ equals s is less than $1/2^{\kappa+\log T}$. Taking the union bound of all T possible queries, the probability is still less than $1/2^\kappa$, so the output distribution changes with $\leq 2^{-\kappa}$ in both cases. Thus, the distributions of $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ are computationally indistinguishable.

$\mathbf{Hyb}_9$: In this hybrid, after checking the evaluator's input sharings, Sim aborts the simulation if the check passes with $\mathsf{CompCheck} = 1$. Recall that $\mathsf{CompCheck} = 1$ only when for more than $t/3$ servers S_j with $\mathsf{Corr}_j = 0$, either of the two cases happen:

- S_j's shares of $[\boldsymbol{v}]_{\times\kappa'}$ are not computed honestly from their committed shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$.
- The first-round OT message of an OT instance for S_j's input shares is not computed honestly from the committed share and the preprocessing data sent to E.

For the first case, if S_j is included in $\mathcal{U}$, the malicious behavior will be detected by G and will cause G to abort, so we do not change the output distribution in this case. For the second case, if S_j is included in $\mathcal{U}$, the only possible reason that G will not abort the protocol is that E commits a different preprocessing data different from what received from $\mathcal{F}_{\mathsf{bitROT}}$ and honestly computes the first-round OT message with the committed data, where the incorrect output from $\mathcal{F}_{\mathsf{bitROT}}$ together with G's output from $\mathcal{F}_{\mathsf{bitROT}}$ is still a pair of valid output from $\mathcal{F}_{\mathsf{bitROT}}$. This happens at most with probability $1/2$ for each invocation of $\mathcal{F}_{\mathsf{bitROT}}$. Therefore, the check can pass with no more than $2^{-\kappa}$ probability when at least κ servers S_j with $\mathsf{Comp}_j = 1$ and $\mathsf{Corr}_j = 0$ are included in $\mathcal{U}$.

Therefore, if $\mathcal{U}$ is uniformly random and sampled after $[\boldsymbol{v}]_{\times\kappa'}$ is sent, then the probability that κ servers S_j among the $t/3$ servers (whose shares of $[\boldsymbol{v}]_{\times\kappa'}$ are not computed honestly from their shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$) are not included in $\mathcal{U}$ can be estimated by the Chernoff bound. More concretely, we add the servers into $\mathcal{U}$ one by one. For each choice of server in $\mathcal{U}$, if the current number of servers $S_j \in \mathcal{U}$ with $\mathsf{Comp}_j = 1$ is smaller than κ, then the probability that this choice of server is among these servers is at least $(t/3 - \kappa)/n$. Note that the probability that κ servers S_j among the $t/3$ servers are chosen in $\mathcal{U}$ is smaller than the probability if each time a random server (allowing repeated choices) is added into $\mathcal{U}$. In this case, each time a server is added, let i be the number of servers in $\mathcal{U}$ after adding this server, then we define:

- $X_i = 1$ if the chosen party is not among a randomly chosen set of $t/3 - \kappa$ servers that are not included in $\mathcal{U}$ before.
- $X_i = 0$ otherwise.

Then, $X_1, \ldots, X_{t/3}$ are independent, with the probability of each $X_i = 1$ be $(t/3 - \kappa)/n$. Let $X = X_1 + \cdots + X_{t/3}$. Then, $\mu := \mathbb{E}(X) = t(t/3 - \kappa)/3n > 4(2\kappa + \log T)$, so by Chernoff Bound:

$$P(X \leq c\kappa) \leq P(X \leq (1 - 3/4)\mu) \leq e^{-\frac{9}{32}\cdot\mu} \leq 2^{-\kappa} \cdot 2^{-\kappa} \cdot T^{-1}.$$

Thus, if $\mathcal{U}$ is uniformly random, the output distribution only changes with probability $1 - (1 - 2^{-\kappa} \cdot 2^{-\kappa} \cdot T^{-1}) \cdot (1 - T^{-\kappa}) < 2^{-\kappa} \cdot T^{-1}$.

Note that when $[\boldsymbol{v}]_{\times\kappa'}$ is not queried by $\mathcal{A}$ to $\mathcal{O}$, then $\mathcal{U}$ is indeed uniformly random. In this case, the distribution only changes with no more than $2^{-\kappa} \cdot T^{-1}$ probability. On the other hand, $\mathcal{A}$ has at most T queries to $\mathcal{O}$, where each time $\mathcal{A}$ needs to fix $[\boldsymbol{v}]_{\times\kappa'}$ and learn whether it can pass the check, so each time

$[\boldsymbol{v}]_{\times\kappa'}$ only passes the check with no more than $2^{-\kappa}\cdot T^{-1}$ probability. Taking the union bound of T queries, the probability that there exists a query of the form $[\boldsymbol{v}]_{\times\kappa'}$ by $\mathcal{A}$ that can pass the check (with more than $t/3$ servers S_j' shares of $[\boldsymbol{v}]_{\times\kappa'}$ are not computed honestly from their shares of $[\boldsymbol{I}_1],\ldots,[\boldsymbol{I}_{k_0}],[\boldsymbol{r}_0]_{\times\kappa'}$ with $\mathsf{Corr}_j=0$) is no more than $T\cdot 2^{-\kappa}\cdot T^{-1}=2^{-\kappa}$. If it does not exist, $\mathcal{A}$ either sends a query he made before, which would not change the distribution, or he will send another sharing $[\boldsymbol{v}]_{\times\kappa'}$ that he has not queried, which changes the output distribution with no more than $2^{-\kappa}$ probability. Therefore, the total probability that the output distribution changes is no more than $2^{-\kappa}+(1-2^{-\kappa})\cdot 2^{-\kappa}=\Theta(1)/2^{\kappa}$. Thus, the distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ are computationally indistinguishable.

$\mathbf{Hyb}_{10}$: In this hybrid, Sim does not follow the protocol to compute $c_1,\ldots,c_n$ and $\sigma_{i,j}$ for each pair of servers (S_i,S_j). Instead, Sim directly samples random strings (with the length as an RO output) as them. Then, the corresponding RO outputs are set to these values.

Note that if $\mathcal{A}$ has not queried $\mathcal{O}$ before they are sent, we do not make any changes. Therefore, the output distribution only changes when one of the queries has been made by $\mathcal{A}$. However, each of the queries contains a uniformly random $(\kappa+\log T)$-bit string, and it is not used to compute any message sent to $\mathcal{A}$ before sending the commitments. Therefore, the probability that the query has been made is no more than $T/2^{\kappa+\log T}=2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ are computationally indistinguishable.

$\mathbf{Hyb}_{11}$: In this hybrid, Sim does not follow the protocol to sample $[\boldsymbol{r}_1]_{\times\kappa'},[\boldsymbol{r}_2]^{(2)}_{\times\kappa'},[\boldsymbol{r}_3]^{(3)}_{\times\kappa'},[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ and compute the 4 sharings in VerSh. Instead, Sim samples the 4 sharings in VerSh randomly and then computes $[\boldsymbol{r}_1]_{\times\kappa'},[\boldsymbol{r}_2]^{(2)}_{\times\kappa'},[\boldsymbol{r}_3]^{(3)}_{\times\kappa'},[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ based on VerSh, the coefficients computed from $\mathcal{O}(\mathsf{GC})$, $[\boldsymbol{x}_1],\ldots,[\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)},\ldots,[\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)},\ldots,[\boldsymbol{z}_{k_3}]^{(3)}$, and $[\boldsymbol{o}_1]^{(2)},\ldots,[\boldsymbol{o}_{k_4}]^{(2)}$. Since $[\boldsymbol{r}_1]_{\times\kappa'},[\boldsymbol{r}_2]^{(2)}_{\times\kappa'},[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$ are uniformly random $\Sigma_{\times\kappa'},\Sigma^{(2)}_{\times\kappa'},\Sigma^{(3)}_{\times\kappa'}$-sharings, and $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ is a uniformly random $\Sigma^{(2)}$-sharing with an all-zero secret, so are the corresponding sharings in VerSh. Therefore, we only change the order of them without changing their distributions. Thus, $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ have the same output distribution.

$\mathbf{Hyb}_{12}$: In this hybrid, Sim does not faithfully emulate $\mathcal{F}_{\mathsf{bitROT}}$ to generate the preprocessing data. Instead, Sim first generates the preprocessing data for E and then samples the preprocessing data for G based on the preprocessing data for E. We only change the order of generating the two parties' outputs from $\mathcal{F}_{\mathsf{bitROT}}$ without changing their distributions. Thus, $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ have the same output distribution.

$\mathbf{Hyb}_{13}$: In this hybrid, Sim does not follow the protocol to generate the garbled circuit. Instead, for each server S_j that is not in the set $\mathcal{C}$ after performing the verification and sampling VerSh, Sim does not follow the protocol to query the random oracle with the queries in Q_j first and then compute the ciphertexts in $\tilde{C}$ with these outputs. Instead, Sim follows $\mathsf{Sim}_{\mathsf{obl}}$ to generate these ciphertexts and emulate $\mathcal{O}$ in the future simulation. Following from the proof of privacy of our garbling scheme, the generated $\tilde{C}$ has the same distribution as computed by following the protocol, and the emulation of $\mathcal{O}$ can be programmed when $\mathcal{A}$ queries an element in Q_j for no more than t of $j\in\{1,\ldots,n\}$. Note that the queries in Q_j are not used for sending any message to $\mathcal{A}$ if $S_j\notin\mathcal{V}$ and $\mathsf{Comp}_j=0$. The distribution only changes when $\mathcal{A}$ queries $t/3$ of them. For the same reason as in $\mathbf{Hyb}_2$, this happens with at most $2^{-\kappa}$ probability. Thus, the distributions of $\mathbf{Hyb}_{13}$ and $\mathbf{Hyb}_{12}$ are computationally indistinguishable.

Note that for the servers that are never included in $\mathcal{C}$, Sim does not need their shares of $[\boldsymbol{r}_1]_{\times\kappa'}$, $[\boldsymbol{r}_2]^{(2)}_{\times\kappa'}$, $[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$, and $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ in the simulation. Sim does not generate them in the future hybrids. In addition, Sim does not need the set $\{\boldsymbol{r}^{(1)}_{\boldsymbol{s}_i\oplus 1},\ldots,\boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}_i\oplus 1}\}^{\mathsf{rec}+n_o}_{i=\mathsf{rec}+1}$ during the simulation, Sim also does not generate them in future hybrids.

$\mathbf{Hyb}_{14}$: In this hybrid, Sim does not sample $\boldsymbol{r}^{S_j}_{\mathsf{commit}}$ and $\boldsymbol{r}^{(i,j)}_{\mathsf{comsh}}$ if $i,j\notin\mathcal{V}$ and check whether the corresponding query to the random oracle has been queried. Note that Sim does not need these random strings except that Sim checks whether these queries are made. Therefore, this only changes the output distribution when one of them is truly queried. Since they are not used for sending any messages to $\mathcal{A}$, each query made by $\mathcal{A}$ only has no more than $1/2^{\log T+\kappa}=T^{-1}\cdot 2^{-\kappa}$ probability to match any of the queries. Taking the union bound for all T possible queries made by $\mathcal{A}$, the probability is still $\leq 2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_{14}$ and $\mathbf{Hyb}_{13}$ are computationally indistinguishable.

$\mathbf{Hyb}_{15}$: In this hybrid, during the simulation of Round 2, corresponding to each bit b from the input shares of servers out of $\mathcal{C}$, Sim does not follow the protocol to compute the second-round OT messages.

More concretely, Sim still follows the protocol to compute $\boldsymbol{x}_b$, but it samples $\boldsymbol{x}_{b\oplus 1}$ randomly. Note that $\boldsymbol{x}_{b\oplus 1}$ is computed by the XOR of an OT message of G and the preprocessing data $\boldsymbol{r}_{\boldsymbol{\beta}\oplus\mathbf{1}}$. Since $\boldsymbol{r}_{\boldsymbol{\beta}\oplus\mathbf{1}}$ is only used in computing $\boldsymbol{x}_{b\oplus 1}$ and it's uniformly random. Thus, $\boldsymbol{x}_{b\oplus 1}$ is also uniformly random, so we randomly generating $\boldsymbol{x}_{b\oplus 1}$ and then computes $\boldsymbol{r}_{\boldsymbol{\beta}\oplus\mathbf{1}}$ will not affect the output distribution. Then, $\boldsymbol{r}_{\boldsymbol{\beta}\oplus\mathbf{1}}$ is not used during the simulation, so Sim no longer generates it in future hybrids. This also does not affect the output distribution. Thus, $\mathbf{Hyb}_{15}$ and $\mathbf{Hyb}_{14}$ have the same output distribution.

$\mathbf{Hyb}_{16}$: In this hybrid, Sim does not follow the protocol to check the garbled outputs received from the evaluator. Instead, Sim directly checks whether $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ matches what Sim generates. As analyzed in the proof of authenticity of our garbling scheme, this only changes the output distribution when E outputs $Y = (\kappa, \{\boldsymbol{s}'_i, \boldsymbol{r}_{\boldsymbol{s}'_i}^{(1)} \ldots, \boldsymbol{r}_{\boldsymbol{s}'_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o})$ such that the check passes, but $\{\boldsymbol{s}'_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ flips $t+1$ bits from $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$. Since Sim does not generate $\{\boldsymbol{r}_{\boldsymbol{s}_i\oplus\mathbf{1}}^{(\alpha)}\}_{\alpha=1}^{\log 40T}$ for $i = \mathsf{rec}+1, \ldots, \mathsf{rec}+n_o$ during the simulation, an adversary who flips $\geq t+1$ bits from $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ can only passes the check with probability $< 2^{-\kappa}$ using a similar analysis as in the proof of authenticity. Thus, the distributions of $\mathbf{Hyb}_{16}$ and $\mathbf{Hyb}_{15}$ are computationally indistinguishable.

$\mathbf{Hyb}_{17}$: In this hybrid, G does not compute his output by himself. Instead, he directly gets his outputs from $\mathcal{F}$, where the evaluator's inputs to $\mathcal{F}$ are reconstructed from the input sharings using error correction. Note that the set $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ received from E match what G generates, we can regard that E honestly run the evaluation algorithm of the garbling scheme. From the correctness of the garbling scheme, G's outputs $\overline{y}_G$ of $\overline{C}$ obtained from the protocol execution is exactly $\mathsf{En}(y_G \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}))$ for G's output y_G from $\mathcal{F}$. The protocol output of G is decoded from $\mathsf{En}(y_G \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}))$ with De. Recall that

$$\mathsf{De}(\mathsf{En}(y_G \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}))) = y_G \oplus \mathsf{mask},$$

the protocol outputs $\mathsf{De}(\overline{y}_G) \oplus \mathsf{mask}$ is the same as the functionality output y_G. Thus, $\mathbf{Hyb}_{17}$ and $\mathbf{Hyb}_{16}$ have the same output distribution.

Note that $\mathbf{Hyb}_{17}$ is the ideal-world scenario, where the statistical difference between any two hybrids $\mathbf{Hyb}_i$ and $\mathbf{Hyb}_{i+1}$ $(i = 0, \ldots, 16)$ is no more than $\Theta(1)/2^\kappa$. Thus, the total difference between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_{17}$ is still at most $\Theta(1)/2^\kappa$, so Π securely realizes $\mathcal{F}$ with computational security when the evaluator E is corrupted.

K.4 Construction from 2-Round OT Protocols

In this section, we show how to construct the 3-round 2PC from 2-round OT protocols with certain properties. An OT protocol runs between an OT sender S and an OT receiver R. We consider the security against probabilistic polynomial-time (PPT) adversaries.

Semi-Malicious 2-Round OT in the CRS Model. We first formalize the requirements on 2-round OTs considered in our construction. In particular, we consider a stronger version of a semi-malicious 2-round OT protocol in the CRS model:

- The input bit of a corrupted OT receiver R can be chosen after receiving the common reference string;
- The input messages of a corrupted OT sender S can be chosen after receiving the common reference string and the first-round OT message;
- We require a stronger computational security, where the advantage of the adversary should be bounded by $2^{-\kappa}$ (which is also the requirement for the 2PC protocol).

Formally, the 2-round OT protocol is defined by four polynomial-time algorithms $(K_{\mathsf{OT}}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$:

- The algorithm K_{OT} takes the security parameter 1^κ as input and outputs the common reference string σ.

- The OT_1 algorithm (which is a probabilistic algorithm) takes as input the common reference string σ and R's input bit b. Then, OT_1 outputs a first-round OT message ots_1 and a state st. R runs OT_1 and sends ots_1 to S in the first round of the OT protocol.
- The OT_2 algorithm (which is a probabilistic algorithm) takes as input the common reference string σ, S's input messages $\boldsymbol{m}_0, \boldsymbol{m}_1 \in \{0,1\}^{\ell_{\mathsf{OT}}}$, and the first-round OT message ots_1. Then, OT_2 outputs a second-round OT message ots_2. S runs OT_2 and sends ots_2 to R in the second round of the OT protocol.
- The OT_3 algorithm takes as input the common reference string σ, the state st, and the second-round OT message ots_2. R runs OT_3 and outputs what OT_3 outputs after receiving ots_2.

We require the protocol to satisfy the following properties (where we omit the security parameter in the inputs of the algorithms for simplicity):

- **Receiver's Security:** When the sender S is corrupted, there exists a PPT simulator Sim_R such that for any $b \in \{0,1\}$, any $m_0, m_1 \in \{0,1\}^{\ell_{\mathsf{OT}}}$, any auxiliary input z, and for any PPT algorithms $m_0'(\mathsf{ots}_1, m_0, m_1, \sigma, z), m_1'(\mathsf{ots}_1, m_0, m_1, \sigma, z), \omega(\mathsf{ots}_1, m_0, m_1, \sigma, z)$, any PPT adversary can at most distinguish
$$\left\{ \begin{array}{l} \sigma \leftarrow K_{\mathsf{OT}}(1^\kappa), \\ (\mathsf{ots}_1, \mathsf{st}) \leftarrow \mathsf{OT}_1(\sigma, b) \end{array} : \sigma, \mathsf{ots}_1, \mathsf{OT}_3(\mathsf{OT}_2(\sigma, \mathsf{ots}_1, m_0', m_1'; \omega), \mathsf{st}) \right\}$$
and
$$\{(\sigma, \mathsf{ots}_1) \leftarrow \mathsf{Sim}_R(1^\kappa) : \sigma, \mathsf{ots}_1, m_b'\}$$
with advantage $2^{-\kappa}$, where ω is the corrupted sender's randomness used in the algorithm OT_2.
- **Sender's Security:** When the receiver R is corrupted, there exist PPT simulators $\mathsf{Sim}_{S,1}, \mathsf{Sim}_{S,2}$ such that for any $b \in \{0,1\}$, any $m_0, m_1 \in \{0,1\}^{\ell_{\mathsf{OT}}}$, any auxiliary input z, and any PPT algorithms $b'(\sigma, b, z), \omega(\sigma, b, z)$, any PPT adversary can at most distinguish
$$\left\{ \begin{array}{l} \sigma \leftarrow K_{\mathsf{OT}}, \\ (\mathsf{ots}_1, \mathsf{st}) \leftarrow \mathsf{OT}_1(\sigma, b'; \omega) \end{array} : \sigma, \mathsf{OT}_2(\sigma, \mathsf{ots}_1, m_0, m_1) \right\}$$
and
$$\{\sigma \leftarrow \mathsf{Sim}_{S,1}(1^\kappa), \mathsf{ots}_2 \leftarrow \mathsf{Sim}_{S,2}(\sigma, b', m_{b'}, \omega) : \sigma, \mathsf{ots}_2\}$$
with advantage $2^{-\kappa}$, where ω is the corrupted receiver's randomness used in the algorithm OT_1.

In particular, we call a 2-round OT protocol that satisfies our requirements a semi-malicious 2-round OT protocol with adapted input choices in the CRS model.

Protocol Construction. Let $(K_{\mathsf{OT}}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ be a semi-malicious 2-round OT protocol with adaptive input choices in the CRS model. We still use T to denote the maximum number of RO queries made by the adversary in our construction. When considering the asymptotic communication complexity, we can take $\log T$ to be any function $f(\kappa) = \omega(\log \kappa)$ to keep the protocol secure against any PPT adversary. Similarly to Section 6, we consider the computational security with advantage $\mathsf{poly}(\kappa)/2^\kappa$. By multiplying a constant $c > 1$ on κ in the construction, the advantage can then be reduced to $< 2^{-\kappa}$. Let $\mathcal{O}$ be a random oracle with a sufficiently large (still $\mathsf{poly}(\kappa)$) output length. The protocol Π runs Π_1 (Figure 38), Π_2 (Figure 39), and Π_3 (Figure 40) in order.

The First Round. The first-round protocol Π_1 is given below in Figure 38. In the first round, the two parties perform local computations to initialize the underlying MPC protocol. Then, the evaluator sends his first-round OT messages together with their authentication information.

Protocol Π_1

The protocol is run by a garbler G and an evaluator E.

1. **Circuit Transformation.** We use the same algorithms En, De as in Garble. For the original circuit $C(x_1, y_1) = (C_G(x_1, y_1), C_E(x_1, y_1))$, where $C_G(x_1, y_1)$ (resp. $C_E(x_1, y_1)$) is the output for G (resp. E), we use $\overline{C}$ to denote the circuit:

$$\begin{aligned}&\overline{C}(x_G \| \mathsf{mask} \| \boldsymbol{r}_{\mathsf{en}}^{(1)} \| \ldots \| \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}, x_E)\\ =&(\mathsf{En}(C_G(x_G, x_E) \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)})), C_E(x_G, x_E)).\end{aligned}$$

2. **Initializing.** Let Π_0' be the underlying protocol based on an $(n = 6t, t = 147(\log T + \kappa), k, \ell)$-LSSS Σ over $\mathbb{F}_2$ that runs among two clients G, E and n servers $S_1, \ldots, S_n$ to compute the circuit $\overline{C}$, where G's input is $x_G \| \mathsf{mask} \| \boldsymbol{r}_{\mathsf{en}}^{(1)} \| \ldots \| \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$, and E's input is x_E. Here x_G, x_E are G, E's inputs to C, respectively.
3. **Generating Preprocessing and Input Data.** E emulates $\mathcal{F}_{\mathsf{input}}^{E}$ to generate his input sharings in Π_0'.
4. **Preparing CRS.** G serves as the sender and E serves as the receiver to run K_{OT} and obtain the common reference string σ of L 2-round OT protocols, where L is the total size of E's input sharings.
5. **Generating Mask Sharing.** Let $\kappa' = n + \kappa$, E samples a random $\Sigma_{\times\kappa'}$-sharing as $[\boldsymbol{r}_0]_{\times\kappa'}$.
6. **Computing First-Round OT Messages.** The parties initialize L instances of OTs with $\ell_{\mathsf{OT}} = \log 40T + 1$, where each instance is prepared for each bit b of the input sharings. For each of the OT protocols, the parties run K_{OT} to obtain the common reference string σ. Then, E computes $(\mathsf{ots}_1, \mathsf{st}) \leftarrow \mathsf{OT}_1(\sigma, b; \omega)$, where ω is the randomly sampled randomness consumed for this instance of OT. Let the set of all the first-round OT messages ots_1 for the bits of server S_j's share of the input sharings (i.e. the inputs to $\mathtt{Circ}^{S_j}$) be $\mathsf{OT1Msg}_j$. For each $j = 1, \ldots, n$, let Input_j be the set containing:
   - inputs to $\mathtt{Circ}^{S_j}$ generated by $\mathcal{F}_{\mathsf{input}}^{E}$;
   - E's randomness ω and the common reference string σ for all instances of OTs where the input bit b is from S_j's shares of E's input sharings.
   - S_j's share of $[\boldsymbol{r}_0]_{\times\kappa'}$.

   E sends $\{\mathsf{OT1Msg}_j\}_{j=1}^n$ to G.
7. **Committing to OT Inputs.** For each $j = 1, \ldots, n$, E samples a random string $\boldsymbol{r}_{\mathsf{comot}}^{S_j} \in \mathbb{F}_2^{\kappa + \log T}$ and queries the random oracle to get $\tau_j = \mathcal{O}(\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j)$. E then sends $\{\tau_j\}_{j=1}^n$ to G.
8. **Generating the Challenge.** Let the number of input Σ-sharings output by $\mathcal{F}_{\mathsf{input}}^{E}$ be k_0. E queries $\mathcal{O}$ with the challenge $s = \{\mathsf{OT1Msg}_j\}_{j=1}^n \| \{\tau_j\}_{j=1}^n$ and lets the first $\kappa' \cdot k_0$ bits of $\mathcal{O}(s)$ be $\theta_1, \ldots, \theta_{k_0} \in \mathbb{F}_{2^{\kappa'}}^{k_0}$.
9. **Computing Linear Combination.** E regards his input sharings $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ as $\Sigma_{\times\kappa'}$-sharings $[\boldsymbol{I}_1]_{\times\kappa'}, \ldots, [\boldsymbol{I}_{k_0}]_{\times\kappa'}$ (by embedding each bit into $\mathbb{F}_{2^{\kappa'}}$). Then, E computes:

$$[\boldsymbol{v}]_{\times\kappa'} = \sum_{i=1}^{k_0} \theta_i [\boldsymbol{I}_i]_{\times\kappa'} + [\boldsymbol{r}_0]_{\times\kappa'}$$

   and sends it to G.
10. **Generating the Challenge for Opening Shares.** E queries $\mathcal{O}$ with $s \| [\boldsymbol{v}]_{\times\kappa'}$ and obtains a subset $\mathcal{U}$ of $t - \kappa$ servers among $S_1, \ldots, S_n$ from the first n bits of $\mathcal{O}(s \| [\boldsymbol{v}]_{\times\kappa'})$. E sends $\mathcal{U}$ to G.
11. **Opening Commitments for OT Inputs.** For each $S_j \in \mathcal{U}$, E sends Input_j and $\boldsymbol{r}_{\mathsf{comot}}^{S_j}$ to G.

Figure 38: The first round of our 2PC from 2-round OTs.

The Second Round. The second-round protocol Π_2 is given in Figure 39. In this round, the garbler verifies the evaluator's OT inputs, responds to the OT messages, sends his garbled circuits, and sends the authentication information for the garbled circuits.

Protocol Π_2

1. **Generating Output Mask.** Let W_O be the number of output wires attached to G. At the beginning of the protocol, the garbler samples a random mask $\mathsf{mask} \in \mathbb{F}_2^{W_O}$ for his output. For each batch of k output wires indexed i attached to G, G samples a random n-bit string $\boldsymbol{r}_{\mathsf{en}}^{(i)}$.
2. **Generating Preprocessing and Input Data.** G emulates $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^{G}$ to generate the preprocessing data and his input sharings in Π_0'.
3. **Checking Evaluator's Input Sharings.** G queries $\mathcal{O}$ with $s = \{\mathsf{OT1Msg}_j\}_{j=1}^n \| \{\tau_j\}_{j=1}^n$ to get $\theta_1, \ldots, \theta_{k_0} \in \mathbb{F}_{2^{\kappa'}}$, and G also queries $\mathcal{O}$ with $s \| [\boldsymbol{v}]_{\times\kappa'}$. Then G verifies whether:
   - The set $\mathcal{U}$ received from E matches $\mathcal{O}(s \| [\boldsymbol{v}]_{\times\kappa'})$.
   - The first-round OT messages in $\{\mathsf{OT1Msg}_j\}_{S_j \in \mathcal{U}}$ are computed honestly from E's OT inputs, the common reference string σ, and the randomness from $\{\mathsf{Input}_j\}_{S_j \in \mathcal{U}}$.
   - $\tau_j = \mathcal{O}(\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j)$ holds for each $S_j \in \mathcal{U}$.
   - The sharing $[\boldsymbol{v}]_{\times\kappa'}$ is a valid $\Sigma_{\times\kappa'}$-sharing.

   If not, G aborts the protocol.
4. **Garbling the Circuit.** Assume that the output wires of $\overline{C}$ attached to G are batched into n_o batches, indexed $j = 1, \ldots, n_o$. For $i = 1, \ldots, \mathsf{rec}$, let the receiver of the i-th reconstruction of $\Sigma^{(2)}$-sharing during Π_0' online phase be R_i, and let the sharing to be reconstructed be $[\boldsymbol{s}_i]^{(2)}$. For each $j = 1, \ldots, n_o$, we let $[\boldsymbol{s}_{\mathsf{rec}+j}]^{(2)}$ to be the output sharing for the j-th batch of output wires attached to G, and we let the receiver $R_{\mathsf{rec}+j}$ to be the client G, as G holds the j-th batch of output wires. Then, G runs Steps 4(a)-4(c),5,6 of Garble and obtains the garbled circuit $\tilde{C}$ corresponding to Π_0'.
5. **Generating Auxiliary Sharings.** G generates a random $\Sigma_{\times\kappa'}$-sharing $[\boldsymbol{r}_1]_{\times\kappa'}$, a random $\Sigma^{(2)}_{\times\kappa'}$-sharing $[\boldsymbol{r}_2]^{(2)}_{\times\kappa'}$, a random $\Sigma^{(3)}_{\times\kappa'}$-sharing $[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$, and a random $\Sigma^{(2)}_{\times\kappa'}$-sharing $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ with an all-zero secret $\boldsymbol{o} = (0, 0, \ldots, 0) \in \mathbb{F}_{2^{\kappa'}}^k$.
6. **Computing Second-Round OT Messages.** For the instance of OT corresponding to a bit b that serves as the i-th input to $\mathtt{Circ}^{S_j}$ generated by $\mathcal{F}_{\mathsf{input}}^{E}$, G parse the ots_1 message from E's message sent in the first round and then computes $\mathsf{ots}_2 = \mathsf{OT}_2(\sigma, \mathsf{ots}_1, X_{i,0}^{(j)}, X_{i,1}^{(j)}; \tilde{\omega})$ as the second-round OT message, where $\tilde{\omega}$ is the randomly sampled randomness. Let the set of all the second-round OT messages ots_2 for the inputs to $\mathtt{Circ}^{S_j}$ be $\mathsf{OT2Msg}_j$. For each $j = 1, \ldots, n$, let Rand_j be the set containing G's randomness $\tilde{\omega}$ for all instances of OTs prepared for the inputs to $\mathtt{Circ}^{S_j}$.
7. **Computing Servers' View.** For each $j \in \mathcal{V}$, let View^{S_j} be the set of:
   - Δ^{S_j};
   - $\lambda_w, k_{w,0}$ for all wires w excluding output wires of XOR/output gates in $\mathtt{Circ}^{S_j}$, all the outputs to S_j from $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^{G}$ excluding S_j's shares of the $\Sigma^{(2)}$-sharings whose secrets are held by other servers;
   - S_j's shares of all $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ corresponding to reconstructions of $\Sigma^{(2)}$-sharings whose receiver is not S_j;
   - all the sharings $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ corresponding to reconstructions of $\Sigma^{(2)}$-sharings whose receiver is S_j;
   - S_j's shares of $[\boldsymbol{r}_1]_{\times\kappa'}$, $[\boldsymbol{r}_2]^{(2)}_{\times\kappa'}$, $[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$, and $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$.
   - The set Rand_j.

   Then, G samples a random string $\boldsymbol{r}_{\mathsf{commit}}^{S_j} \in \mathbb{F}_2^{\kappa+\log T}$ and queries the random oracle to get $c_j = \mathcal{O}(\mathsf{View}^{S_j} \| \boldsymbol{r}_{\mathsf{commit}}^{S_j} \| j)$. Moreover, for each pair of servers (S_i, S_j), let all the shares for S_j from $\mathcal{F}_{\mathsf{prep}}$ of the sharings whose secrets are held by S_i form a set $\mathsf{Sh}_{i,j}$. Then, G samples a random string $\boldsymbol{r}_{\mathsf{comsh}}^{(i,j)} \in \mathbb{F}_2^{\kappa+\log T}$ and queries the random oracle to get $\sigma_{i,j} = \mathcal{O}(\mathsf{Sh}_{i,j} \| \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)} \| i \| j)$.
8. **Computing Coefficients.** Let $\tilde{C} = ((F_1, L_1, d_1), \ldots, (F_n, L_n, d_n), F_{\mathsf{ct}})$, and let

$$\mathsf{GC} = (\tilde{C}, \{\mathsf{OT2Msg}_j\}_{j=1}^n, c_1, \ldots, c_n, \{\sigma_{i,j}, \sigma_{j,i}\}_{1 \le i < j \le n}).$$

Recall that $\kappa' = n + \kappa$. Let the number of Σ-sharings (resp. $\Sigma^{(2)}$-sharings, $\Sigma^{(3)}$-sharings, $\Sigma^{(2)}$-sharings with all-zero secrets) generated by G while emulating $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}^{G}_{\mathsf{input}}$ and during Step 4 of Garble be k_1 (resp. k_2, k_3, k_4). G queries $\mathcal{O}$ with the challenge GC and lets the first $\kappa'(k_1+k_2+k_3+k_4)$ bits of $\mathcal{O}(\mathsf{GC})$ be $(\alpha_1, \ldots, \alpha_{k_1}, \beta_1, \ldots, \beta_{k_2}, \gamma_1, \ldots, \gamma_{k_3}, \eta_1, \ldots, \eta_{k_4})$ in $\mathbb{F}_{2^{\kappa'}}^{k_1+k_2+k_3+k_4}$.

9. **Generating Verification Sharings.** Let the k_1 Σ-sharings counted in the last step be $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, let the k_2 $\Sigma^{(2)}$-sharings be $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, let the k_3 $\Sigma^{(3)}$-sharings be $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, and let the k_4 $\Sigma^{(2)}$-sharings with all-zero secrets be $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$. G regards them as $\Sigma_{\times\kappa'}, \Sigma^{(2)}_{\times\kappa'}, \Sigma^{(3)}_{\times\kappa'}$-sharings (by embedding each bit into $\mathbb{F}_{2^{\kappa'}}$). Then, G computes VerSh as the set of
$$\Sigma_{i=1}^{k_1} \alpha_i [\boldsymbol{x}_i]_{\times\kappa'} + [\boldsymbol{r}_1]_{\times\kappa'}, \qquad \Sigma_{i=1}^{k_2} \beta_i [\boldsymbol{y}_i]^{(2)}_{\times\kappa'} + [\boldsymbol{r}_2]^{(2)}_{\times\kappa'},$$
$$\Sigma_{i=1}^{k_3} \gamma_i [\boldsymbol{z}_i]^{(3)}_{\times\kappa'} + [\boldsymbol{r}_3]^{(3)}_{\times\kappa'}, \text{ and } \Sigma_{i=1}^{k_4} \eta_i [\boldsymbol{o}_i]^{(2)}_{\times\kappa'} + [\boldsymbol{o}]^{(2)}_{\times\kappa'}.$$
10. **Generating the Challenge for Opening Views.** G queries $\mathcal{O}$ with $\mathsf{GC}\|\mathsf{VerSh}$ and lets the first n bits of the output denote a set $\mathcal{V}$ of $t/3$ indices in $\{1, \ldots, n\}$.
11. **Sending the Garbled Circuit.** G sends $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}^{S_j}_{\mathsf{commit}}\}_{j\in\mathcal{V}}, \{\mathsf{Sh}_{i,j}, \boldsymbol{r}^{(i,j)}_{\mathsf{comsh}}, \mathsf{Sh}_{j,i}, \boldsymbol{r}^{(j,i)}_{\mathsf{comsh}}\}_{i\in\mathcal{V}, j\in\{1,\ldots,n\}\setminus\{i\}})$ to E.

Figure 39: The second round of our 2PC from 2-round OTs.

The Third Round. The third-round protocol Π_3 is given in Figure 40. In this round, the evaluator verifies the authentication information of the garbled circuit and second-round OT messages. Finally, the evaluator obtains his own circuit outputs and sends the GC outputs to the garbler.

Protocol Π_3

1. **Verifying the Garbled Circuit.** For each instance of OT, the evaluator runs $\mathsf{OT}_3(\mathsf{ots}_2, \mathsf{st})$ and then obtains all the input wire labels. The evaluator E computes $(\alpha_1, \ldots, \alpha_{k_1}, \beta_1, \ldots, \beta_{k_2}, \gamma_1, \ldots, \gamma_{k_3}, \eta_1, \ldots, \eta_{k_4})$ from $\mathcal{O}(\mathsf{GC})$ checks whether the following hold:
   - The 4 sharings in VerSh are valid $\Sigma_{\times\kappa'}$, $\Sigma^{(2)}_{\times\kappa'}$, $\Sigma^{(3)}_{\times\kappa'}$-sharings and a $\Sigma^{(2)}_{\times\kappa'}$-sharing with an all-zero secret.
   - $c_j = \mathcal{O}(\mathsf{View}^{S_j}\|\boldsymbol{r}^{S_j}_{\mathsf{commit}}\|j)$ for each $j \in \mathcal{V}$, and $\sigma_{i,j} = \mathcal{O}(\mathsf{Sh}_{i,j}\|\boldsymbol{r}^{(i,j)}_{\mathsf{comsh}}\|i\|j)$, $\sigma_{j,i} = \mathcal{O}(\mathsf{Sh}_{j,i}\|\boldsymbol{r}^{(j,i)}_{\mathsf{comsh}}\|j\|i)$ for each $j \in \mathcal{V}, i \in \{1, \ldots, n\}\setminus\{j\}$.
   - For the output $\Sigma^{(2)}$-sharings from $\mathcal{F}_{\mathsf{prep}}$ to $j \in \mathcal{V}$ where S_j holds the secrets, the shares from $\{\mathsf{Sh}_{j,i}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$ form valid sharings, and the secret in View^{S_j} matches the sharings.
   - $\tilde{C}, \{\mathsf{OT2Msg}_j\}_{j\in\mathcal{V}}, d^E_{\mathsf{prep}}, \mathsf{VerSh}$ match $\{\mathsf{View}^{S_j}\}_{j\in\mathcal{V}}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i\in\mathcal{V}, j\in\{1,\ldots,n\}\setminus\{i\}}$. More concretely, the following elements in $\tilde{C}, \{\mathsf{OT2Msg}_j\}_{j\in\mathcal{V}}, d^E_{\mathsf{prep}}, \mathsf{VerSh}$ can be computed from $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}$:
     - (F_j, L_j, d_j) and the ciphertexts in F_{ct} generated in Step 6 of Garble for $R_i = S_j$.
     - The wire labels for S_j's shares of E's input sharings.
     - S_j's shares of the 4 sharings in VerSh.
     - The set $\mathsf{OT2Msg}_j$ of second-round OT messages.
   - The obtained wire labels via OT_3 for the input shares of S_j matches the labels in View^{S_j}.
   - $\mathcal{V}$ matches $\mathcal{O}(\mathsf{GC}\|\mathsf{VerSh})$.
2. **Evaluation.** E follows the evaluation algorithm Eval to evaluate the garbled circuit, except that E uses error correction to reconstruct the secrets (see Section H for the detailed evaluation process).
3. **Sending the Output.** After evaluating the garbled circuits, E reconstructs his own outputs from the output sharings via error correction and sends $\{\boldsymbol{s}_i, \boldsymbol{r}^{(1)}_{\boldsymbol{s}_i}, \ldots, \boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}_i}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ to G.
4. **Decoding the Output.** G runs the Decode to decode his output $\overline{y}_G$ of $\overline{C}$. Then, G checks whether $\overline{y}_G$ is a valid output of En. If not, G aborts the protocol. Otherwise, G computes his outputs of C by $\mathsf{De}(\overline{y}_G) \oplus \mathsf{mask}$.

Figure 40: The third round of our 2PC from 2-round OTs.

Taking $\log T$ to be any function $f(\kappa) = \omega(\log \kappa)$, we obtain the following result.

Theorem 7 *For any function $f(\kappa) = \omega(\log \kappa)$ assuming a semi-malicious 2-round OT protocol (with message length $O(f(\kappa))$) with adaptive input choices in the CRS model (that achieves the computational security with advantage $2^{-\kappa}$ with communication cost $\mathsf{CC}_{\mathsf{OT}}(\kappa)$) and a random oracle, there is a 3-round maliciously secure maliciously secure two-party computation protocol for any Boolean circuit C with communication of $O(|C|f(\kappa) + D\kappa^2 f(\kappa) + \mathsf{poly}(\kappa) + W_I \mathsf{CC}_{\mathsf{OT}})$ bits against any PPT adversary, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the computational security parameter. The protocol achieves computational security with advantage $2^{-\kappa}$.*

When G has no output, Steps 3 and 4 in Π_3 are not required. Then, Π_3 only includes the evaluator's local computation, which gives a 2-round protocol, i.e. a non-interactive secure computation (NISC) protocol, with the same communication complexity.

Theorem 4 *For any function $f(\kappa) = \omega(\log \kappa)$, assuming a semi-malicious 2-round OT protocol (with message length $O(f(\kappa))$) with adaptive input choices in the CRS model (that achieves the computational security with advantage $2^{-\kappa}$ with $\mathsf{CC}_{\mathsf{OT}}(\kappa)$ communication cost) and a random oracle, there is a maliciously secure non-interactive secure computation protocol for any Boolean circuit C with communication of $O(|C|f(\kappa) + D\kappa^2 f(\kappa) + \mathsf{poly}(\kappa) + W_I \mathsf{CC}_{\mathsf{OT}}(O(\kappa)))$ bits against any probabllistic polynomial-time adversary, where $|C|$ is the circuit size, D is the circuit depth, W_I is the number of input wires, and κ is the computational security parameter. The protocol achieves computational security with advantage $2^{-\kappa}$.*

K.5 Security Proof for the Protocol from 2-Round OTs

We now prove the security of our 2PC protocol Π presented in Section K.4. For simplicity, we say two distributions are computationally indistinguishable if any PPT adversary $\mathcal{A}$ cannot distinguish them with an advantage larger than $\mathsf{poly}(\kappa)/2^{\kappa}$.

We prove the security by constructing an ideal adversary Sim. Then we will show that the output in the ideal world is computationally indistinguishable from that in the real world using hybrid arguments. Note that $\mathcal{A}$ may corrupt either the garbler G or the evaluator E, so we prove the security under the two cases separately.

Security against a Corrupted Garbler. When G is corrupted, we construct the ideal adversary Sim below. We separate the description of Sim into the simulators $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ for the 3 rounds, respectively.

Simulator Sim_1

1. Sim initializes $\mathsf{ShCheck} = \mathsf{CompCheck} = 0$ and $\mathsf{Comp}_1 = \cdots = \mathsf{Comp}_n = 0$.
2. Sim follows the protocol to initialize the protocol Π'_0.
3. For each $j = 1, \ldots, n$, Sim samples τ_j randomly and sends it to G on behalf of E.
4. Sim samples a random $\Sigma_{\times\kappa'}$-sharing as $[\boldsymbol{v}]_{\times\kappa'}$ and samples a random subset of $t - \kappa$ servers as $\mathcal{U}$.
5. For each instance of OT for the inputs to $\mathtt{Circ}^{S_j}$ for each $S_j \in \mathcal{U}$, Sim runs K_{OT} to generate the common reference string σ and then randomly samples S_j's shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ together with the randomness ω for generating the first-round OT message (such that $\{\mathsf{Input}_j\}_{S_j \in \mathcal{U}}$ has been generated except S_j's share of $[\boldsymbol{r}_0]_{\times\kappa'}$). Then, Sim follows the protocol compute $\mathsf{OT1Msg}_j$. For each instance of OT for the inputs to $\mathtt{Circ}^{S_j}$ for each $S_j \notin \mathcal{U}$, Sim runs the simulator Sim_R to generate σ and the first-round OT message. Then, Sim obtains $\mathsf{OT1Msg}_j$. Finally, Sim sends $\{\mathsf{OT1Msg}_j\}_{j=1}^n$ to G on behalf of E.
6. For each $j = 1, \ldots, n$, Sim samples the RO output τ_j randomly. Then, Sim sends $\{\tau_j\}_{j=1}^n$ to G on behalf of E. Whenever the first $\kappa + 2\log T$ bits of the sampled outputs of $\mathcal{O}$ for two different queries are the same, Sim aborts the simulation.
7. Sim follows the protocol to emulate the random oracle and compute the coefficients $\theta_1, \ldots, \theta_{k_0}$. Sim

computes their shares of $[\boldsymbol{r}_0]_{\times\kappa'}$ by

$$[\boldsymbol{r}_0]_{\times\kappa'} = [\boldsymbol{v}]_{\times\kappa'} - \sum_{i=1}^{k_0} \theta_i [\boldsymbol{I}_i]_{\times\kappa'}.$$

8. Sim samples a random $(\kappa + \log T)$-bit string as $\boldsymbol{r}_{\mathsf{comot}}^{S_j}$ for each $S_j \in \mathcal{U}$. Sim then sends $\{\mathsf{Input}_j, \boldsymbol{r}_{\mathsf{comot}}^{S_j}\}_{S_j \in \mathcal{U}}$ to G on behalf of E. For each $S_j \in \mathcal{U}$, Sim sets the RO output for the query $\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j$ to τ_j.

Figure 41: The simulator for the first round when G is corrupted.

Simulator Sim_2

1. For each $j = 1, \ldots, n$, Sim follows the protocol to compute (F_j, L_j, d_j) and the ciphertexts in F_{ct} generated in Step 6 of Garble for $R_i = S_j$ from $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$ received by $\mathcal{O}$.
2. Sim emulates $\mathcal{O}$ to receive $\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}$ for each $j = 1, \ldots, n$ and $\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}$ for each pair of servers (S_i, S_j). Sim honestly emulates $\mathcal{O}$ to send the outputs to G. Then, Sim retrieve the corresponding committed $(\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j})$ for each $j = 1, \ldots, n$ and $(\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)})$ for each pair of servers (S_i, S_j) from the RO queries that match the commitments. If such committed inputs and randomness are not found from the RO queries for some commitment, Sim sets the committed values corresponding to this commitment to $\perp$.
3. Sim honestly emulates $\mathcal{O}$ to receive GC from G and outputs s' to G. Similarly, Sim honestly emulates $\mathcal{O}$ to receive $\mathsf{GC} \| \mathsf{VerSh}$ from G and outputs $\mathcal{V}$ to G.
4. Sim receives $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}\}_{j \in \mathcal{V}}, \{\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}, \mathsf{Sh}_{j,i}, \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}\}_{i \in \mathcal{V}, j \in \{1,\ldots,n\} \setminus \{i\}})$ from G on behalf of E. Then for each $j = 1, \ldots, n$, Sim checks whether:
   - (F_j, L_j, d_j) and the ciphertexts in F_{ct} for $R_i = S_j$ match what he computed in the last step.
   - The wire labels for S_j's shares of E's input sharings and the randomness set Rand_j in View^{S_j}, the common reference strings, and the first-round OT messages match the second-round OT messages in $\mathsf{OT2Msg}_j$.
   - $\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$ received in this step match what he received while emulating $\mathcal{O}$ in last two steps if $j \in \mathcal{V}$.
   - For the output $\Sigma^{(2)}$-sharings from $\mathcal{F}_{\mathsf{prep}}$ to S_j where S_j holds the secrets, the shares from $\{\mathsf{Sh}_{j,i}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$ form valid sharings, and the secret in View^{S_j} matches the sharings.
   - S_j's shares of the four sharings in VerSh are computed honestly from the shares in $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}\}_{i \in \{1,\ldots,n\} \setminus \{j\}}$.

   If not, Sim sets $\mathsf{Comp}_j = 1$.
5. If for more than $t/2$ indices $j \in \{1, \ldots, n\}$, it satisfies $\mathsf{Comp}_j = 1$, then Sim sets $\mathsf{CompCheck} = 1$.
6. If $\mathsf{CompCheck} = 0$, for each j with $\mathsf{Comp}_j = 0$, Sim obtains S_j's shares of $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, and $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$ from what G sends to $\mathcal{O}$ in the first round. Sim then checks whether these servers' shares of these sharings are all from valid $\Sigma/\Sigma^{(2)}/\Sigma^{(3)}$-sharings and $\Sigma^{(2)}$-sharings with all-zero secrets. If not, Sim sets $\mathsf{ShCheck} = 1$.

Figure 42: The simulator for the second round when G is corrupted.

Simulator Sim_3

1. Sim follows the protocol to verify the garbled circuit. If the check does not pass, Sim aborts the protocol on behalf of E.
2. If $\mathsf{ShCheck} = 1$ or $\mathsf{CompCheck} = 1$, Sim aborts the simulation.
3. Sim reconstructs $\mathsf{mask}, \boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$ from the outputs of $\mathcal{F}_{\mathsf{input}}^{G}$ (received by $\mathcal{O}$) using error correction. Then, Sim uses then and the outputs of C for G to compute G's outputs of $\overline{C}$, i.e. $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$. Sim then

computes the corresponding $\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)} = \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)} + (\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}) * \boldsymbol{s}_i$ for each $\alpha = 1, \ldots, \log 40T$, where $\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}, \boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)}$ are reconstructed from the sharings $[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]$ received by $\mathcal{O}$. Then, Sim sends $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ to G on behalf of E.

4. Sim outputs what $\mathcal{A}$ outputs.

Figure 43: The simulator for the third round when G is corrupted.

We construct the following hybrids:

$\mathbf{Hyb}_0$: In this hybrid, Sim gets E's inputs and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, Sim additionally sets $\mathsf{ShCheck} = \mathsf{CompCheck} = 0$ and $\mathsf{Comp}_1 = \cdots = \mathsf{Comp}_n = 0$ at the beginning of the simulation. This does not affect the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, while emulating the random oracle $\mathcal{O}$, Sim aborts the simulation if the first $\kappa + 2\log T$ bits of the RO outputs are the same for two different queries. Then, for each commitment generated by $\mathcal{O}$ from G, Sim can extract the committed inputs and randomness from the queries made by G. If such a query that matches the commitment is not found, Sim sets the committed inputs and randomness to $\bot$. Note that the RO output for each query is sampled randomly, so the probability that the first $\kappa + 2\log T$ bits of two outputs are the same is $1/2^{-(\kappa+2\log T)}$. Taking the union bound of all $< T^2$ possible pairs of queries, the probability is still less than $2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are computationally indistinguishable.

$\mathbf{Hyb}_3$: In this hybrid, Sim does not follow the protocol to sample $[\boldsymbol{r}_0]_{\times\kappa'}$ and compute $[\boldsymbol{v}]_{\times\kappa'}$. Instead, Sim samples $[\boldsymbol{v}]_{\times\kappa'}$ first and then computes $[\boldsymbol{r}_0]_{\times\kappa'}$ by

$$[\boldsymbol{r}_0]_{\times\kappa'} = [\boldsymbol{v}]_{\times\kappa'} - \sum_{i=1}^{k_0} \theta_i [\boldsymbol{I}_i]_{\times\kappa'}.$$

Since $[\boldsymbol{r}_0]_{\times\kappa'}$ is a uniformly random $\Sigma_{\times\kappa'}$-sharing, so is $[\boldsymbol{v}]_{\times\kappa'}$. Therefore, we only change the order of them without changing their distribution. Thus, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ have the same output distribution.

$\mathbf{Hyb}_4$: In this hybrid, Sim does not follow the protocol to emulate the random oracle and compute $\{\tau_j\}_{j=1}^n$. Instead, Sim directly samples them randomly and then sets the RO outputs for the corresponding inputs $\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j$.

Note that when $\mathcal{A}$ does not query the random oracle with $\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j$, we do not change anything. Thus, the distribution only changes when $\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j$ is queried before the first-round message is sent. Since $\boldsymbol{r}_{\mathsf{comot}}^{S_j} \in \{0,1\}^{\kappa+\log T}$, each query made by $\mathcal{A}$ matches $\mathsf{Input}_j \| \boldsymbol{r}_{\mathsf{comot}}^{S_j} \| j$ with a probability $\leq 2^{-(\kappa+\log T)}$. Taking the union bound of at most T queries, the probability is still $\leq 2^{-\kappa}$. Thus, the output distributions of $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ are computationally indistinguishable.

$\mathbf{Hyb}_5$: In this hybrid, Sim does not follow the protocol to emulate the random oracle and compute $\mathcal{U}$. Instead, Sim directly samples it randomly and then sets the RO outputs for the corresponding input $\mathcal{O}(s \| [\boldsymbol{v}]_{\times\kappa'})$.

Note that when $\mathcal{A}$ does not query the random oracle with $s \| [\boldsymbol{v}]_{\times\kappa'}$, we do not change anything. Thus, the distribution only changes when $s \| [\boldsymbol{v}]_{\times\kappa'}$ is queried before the first-round message is sent. Since $s \| [\boldsymbol{v}]_{\times\kappa'}$ contains $\{\tau_j\}_{j=1}^n$, where each τ_j is a random string of length $> \kappa + \log T$, for the same reason as in $\mathbf{Hyb}_4$, the output distributions of $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ are computationally indistinguishable.

$\mathbf{Hyb}_6$: In this hybrid, while generating $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$, Sim first generates the shares for servers in $\mathcal{U}$ and then samples the other servers' shares based on these shares and the secrets. Note that $|\mathcal{U}| < t$, the shares for servers in $\mathcal{U}$ are independent of the secrets. Therefore, we only change the order of the generations for different servers' shares without changing their distributions. Thus, $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ have the same output distribution.

$\mathbf{Hyb}_7$: In this hybrid, after receiving $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}\}_{j\in\mathcal{V}})$ from G on behalf of E, for each $j = 1, \ldots, n$, Sim additionally checks whether:

- (F_j, L_j, d_j) and the ciphertexts in F_{ct} for $R_i = S_j$ match what he computed in the last step.
- The wire labels for S_j's shares of E's input sharings and the randomness set Rand_j in View^{S_j}, the random reference strings, and the first-round OT messages match the second-round OT messages in $\mathsf{OT2Msg}_j$.
- $\mathsf{View}^{S_j}, \boldsymbol{r}^{S_j}_{\mathsf{commit}}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i\in\{1,\dots,n\}\setminus\{j\}}$ received in this step match what he received while emulating $\mathcal{O}$ in last two steps if $j \in \mathcal{V}$.
- For the output $\Sigma^{(2)}$-sharings from $\mathcal{F}_{\mathsf{prep}}$ to S_j where S_j holds the secrets, the shares from $\{\mathsf{Sh}_{j,i}\}_{i\in\{1,\dots,n\}\setminus\{j\}}$ form valid sharings, and the secret in View^{S_j} matches the sharings.
- S_j's shares of the four sharings in VerSh are computed honestly from the shares in $\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}\}_{i\in\{1,\dots,n\}\setminus\{j\}}$.

If not, Sim sets $\mathsf{Comp}_j = 1$. If for more than $t/2$ indices $j \in \{1,\dots,n\}$, it satisfies $\mathsf{Comp}_j = 1$, then Sim sets $\mathsf{CompCheck} = 1$. This does not affect the output distribution. Thus, $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have the same output distribution.

$\mathbf{Hyb}_8$: In this hybrid, after the check on the servers' computation, if $\mathsf{CompCheck} = 0$, for each j with $\mathsf{Comp}_j = 0$, Sim additionally computes S_j's shares of $[\boldsymbol{x}_1], \dots, [\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)}, \dots, [\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)}, \dots, [\boldsymbol{z}_{k_3}]^{(3)}$, $[\boldsymbol{o}_1]^{(2)}, \dots, [\boldsymbol{o}_{k_4}]^{(2)}$ from what $\mathcal{O}$ receives. Sim then checks whether these servers' shares of these sharings are all from valid $\Sigma/\Sigma^{(2)}/\Sigma^{(3)}$-sharings and valid $\Sigma^{(2)}$-sharings with all-zero secrets. If not, Sim sets $\mathsf{ShCheck} = 1$. This does not affect the output distribution. Thus, $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ have the same output distribution.

$\mathbf{Hyb}_9$: In this hybrid, Sim does not follow the protocol to compute the common reference string and E's first-round OT messages for the OT instances corresponding to S_j's input shares for $S_j \notin \mathcal{U}$. Instead, Sim runs the simulator Sim_R for the OT protocol to generate the common reference string in these OT instances together with the first-round OT messages. Sim delays the generation of his input bits before the evaluation, and local randomness for computing these first-round OT messages is no longer generated in future hybrids. Note that the input bits of these OT protocols are not used to compute any message sent to $\mathcal{A}$. Thus, the sender's security of the OT protocol ensures that the first-round OT messages generated by the simulator and by following the protocol are indistinguishable to $\mathcal{A}$, and local randomness for computing these first-round OT messages are not used in the simulation, so not generating them will not affect the output distribution. We argue that the distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ are computationally indistinguishable by constructing hybrids $\mathbf{Hyb}_{9.1}, \dots, \mathbf{Hyb}_{9.N_{\mathsf{OT}}}$, where N_{OT} is the number of executions of the OT protocol.

$\mathbf{Hyb}_{9.i}$: In this hybrid, Sim does not follow the protocol to compute the common reference string and E's first-round OT messages for the i-th OT instance corresponding to S_j's input shares for $S_j \notin \mathcal{U}$. Instead, Sim runs the simulator Sim_R for the OT protocol to generate the common reference string in the i-th OT together with the first-round OT message. Let $\mathbf{Hyb}_{9.0} = \mathbf{Hyb}_8$, we argue that the distributions of $\mathbf{Hyb}_{9.i}$ and $\mathbf{Hyb}_{9.i-1}$ are computationally indistinguishable.

For the sake of contradiction, assume that $\mathbf{Hyb}_{9.i}$ and $\mathbf{Hyb}_{9.i-1}$ are distinguishable, so there exists an adversary $\mathcal{A}$ in our model that can distinguish the output distributions before and after we change σ, ots_1 to the simulated messages (the outputs m_b of the OT protocols are changed coordinately). Now we construct an adversary $\mathcal{A}'$ in our model that breaks the receiver's security of the i-th OT protocol. Concretely, $\mathcal{A}'$, with the auxiliary input of all the messages received in the first round together with the common reference strings excluding those in the i-th OT protocol, receives a string from the challenger, which is $\sigma, \mathsf{ots}_1, m_b$ either computed from the real protocol or the simulator. Then $\mathcal{A}'$ runs Sim with the change that $\sigma, \mathsf{ots}_1, m_b$ are replaced by the string received from the challenger. Finally, $\mathcal{A}'$ uses $\mathcal{A}$ to distinguish whether the challenge is computed from the real protocols or the simulators. Note that the advantage of $\mathcal{A}'$ is the same as that of $\mathcal{A}$, which breaks the security of the i-th OT protocol. This leads to a contradiction.

Thus, $\mathbf{Hyb}_{9.i}$ and $\mathbf{Hyb}_{9.i-1}$ have the same output distribution.

Note that $\mathbf{Hyb}_{9.N_{\mathsf{OT}}}$ is exactly $\mathbf{Hyb}_9$, so the distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ are computationally indistinguishable.

$\mathbf{Hyb}_{10}$: In this hybrid, if the verification of E passes with $\mathsf{ShCheck} = 1$, Sim aborts the simulation. This only changes the distribution when there exists a set of $\le t/2$ servers where each server S_j in them

satisfies $\mathsf{Comp}_j = 1$, such that the other parties' shares of a Σ or $\Sigma^{(2)}$ or $\Sigma^{(3)}$-sharing or a $\Sigma^{(2)}$-sharing with an all-zero secret committed by G in the first round are not from a valid sharing, but they pass the check for the sharings. Note that $\geq n - t/2$ servers' shares of the sharings in VerSh must be computed from the committed shares, as $\mathsf{Comp} = 0$ for $\geq n - t/2$ parties. Assume the $\geq n - t/2$ servers' shares of a Σ-sharing $[\boldsymbol{x}_i]$ are not from a valid Σ-sharing, there is only one possible $\alpha_i \in \mathbb{F}_{2^{\kappa'}}$ that can make the sharings in VerSh valid when other coefficients are fixed. Otherwise, if there is an $\alpha_i' \neq \alpha_i$ that makes the $\geq n - t/2$ servers' shares of the sharings in VerSh valid, then the $n - t/2$ servers' shares of $(\alpha_i - \alpha_i') \cdot [\boldsymbol{x}_i]$ must comes from a valid Σ-sharing, which conflicts with the fact that the $\geq n - t/2$ servers' shares of $[\boldsymbol{x}_i]$ are invalid. If these servers' shares of a $\Sigma^{(2)}$-sharing or a $\Sigma^{(3)}$-sharing or a $\Sigma^{(2)}$-sharing of an all-zero secret are invalid, the case is similar. Therefore, if all coefficients are uniformly randomly sampled after the servers' views are committed, the output distribution only changes with probability $\leq 2^{-\kappa'}$ when the set of $n - t/2$ parties is fixed. Taking the union bound of $< 2^n$ all such sets of servers, the probability is still $\leq 2^{-(\kappa' - n)} = 2^{-\kappa}$.

Recall that the coefficients are determined by $\mathcal{O}(\mathsf{GC})$, where GC contains all the commitments for the servers' views. When $\mathcal{A}$ never queries GC before the servers' views are committed, the coefficients are uniformly random to him, and the distribution only changes with probability $\leq 2^{-\kappa}$. On the other hand, the commitments are uniformly random RO outputs (whose length $> \kappa + \log T$) before $\mathcal{A}$ queries the random oracle to generate them. Therefore, the probability that $\mathcal{A}$ queries $\mathcal{O}$ with GC before receiving the commitments is still no more than $2^{-\kappa}$, so the output distribution changes with $\leq 2^{-\kappa}$ in both cases. Thus, the distributions of $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ are computationally indistinguishable.

$\mathbf{Hyb}_{11}$: In this hybrid, if the verification of E passes with $\mathsf{CompCheck} = 1$, Sim aborts the simulation. This only changes the distribution when for $> t/2$ servers in the multiparty garbling process, G does not compute the garbled circuit from the committed views of them, or the committed labels of their output of $\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{input}}^G$ together with the randomness do not match the second-round OT messages. For all the cases where G does not compute the garbling of a server S_j from the committed view View^{S_j}, E can detect it if $j \in \mathcal{V}$, so the distribution only changes when G does not compute the garbled circuit from the committed views of $> t/2 > 70(\kappa + \log T)$ servers.

Recall that the set $\mathcal{V}$ is determined by $\mathcal{O}(\mathsf{GC} \| \mathsf{VerSh})$. That is to say, $\mathcal{V}$ is not determined before GC and VerSh are determined. Fixing GC and VerSh, the probability that $\mathcal{V}$ does not include the $> 18(\kappa + \log T)$ indices is no more than

$$\begin{aligned}
&\frac{n - 70(\kappa + \log T)}{n} \cdot \frac{n - 70(\kappa + \log T) - 1}{n - 1} \cdot \cdots \cdot \frac{n - 70(\kappa + \log T) - t/3 + 1}{n - t/3 + 1} \\
&< \left(\frac{n - 70(\kappa + \log T)}{n}\right)^{t/3} < \left(\frac{11}{12}\right)^{49(\kappa + \log T)} < 2^{-(\kappa + \log T)} = T^{-1} \cdot 2^{-\kappa}.
\end{aligned}$$

Note that $\mathcal{A}$ can query $\mathcal{O}$ for T times, where each time $\mathcal{A}$ can fix a pair of GC and VerSh and learn whether this pair can pass the check. Taking the union bound of them, the probability of $\mathcal{A}$ correctly obtaining a pair of GC and VerSh matching the condition that will make the output distribution change is no more than $2^{-\kappa}$. Otherwise, either the pair of GC and VerSh sent by G does not match the condition, or the query $\mathsf{GC} \| \mathsf{VerSh}$ is not made by the adversary $\mathcal{A}$. If the query is not made, $\mathcal{V}$ is random to $\mathcal{A}$, and the probability that the distribution changes is no more than $T^{-1} \cdot 2^{-\kappa}$. In conclusion, the output distribution only changes with probability of $\Theta(1)/2^{\kappa}$. Thus, the distributions of $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ are computationally indistinguishable.

$\mathbf{Hyb}_{12}$: In this hybrid, Sim does not follow the protocol to evaluate the garbled circuit on behalf of E. Instead, Sim only follows that protocol to evaluate the garbled subcircuits for the $\geq n - t$ servers S_j with $\mathsf{Comp}_j = 0$ whose shares of the $\Sigma, \Sigma^{(2)}, \Sigma^{(3)}$-sharings and the $\Sigma^{(2)}$-sharings with all-zero secrets are all from valid sharings. The secrets of $\Sigma^{(2)}, \Sigma^{(3)}$-sharings required to reconstruct in the evaluation process are reconstructed from the shares for these servers. Note that from the correctness of our garbling scheme, these servers' shares of the $\Sigma^{(2)}, \Sigma^{(3)}$-sharings reconstructed in the evaluation process are valid. The only difference from the original evaluation process is that Sim does not evaluate the sub-circuits for the other servers, and he does not use error correction to reconstruct the secrets for these sharings. Since the $\Sigma^{(2)}, \Sigma^{(3)}$-sharings have error-correction properties for up to t errors, no matter how the other t servers' garbled sub-circuits are

evaluated, the secrets of these sharings are fixed, so we do not change any message sent to G. Thus, $\mathbf{Hyb}_{12}$ and $\mathbf{Hyb}_{11}$ have the same output distribution.

$\mathbf{Hyb}_{13}$: In this hybrid, Sim additionally computes the valid shares and garbled sub-circuits for the $\leq t$ servers, excluding the $\geq n-t$ servers taken in the last hybrid, from the committed views to them and simulates the whole evaluation process honestly using these garbled sub-circuits. Since the secrets of the $\Sigma^{(2)}, \Sigma^{(3)}$-sharings reconstructed in the evaluation process have already been determined by the other $n-t$ parties, we do not change any message sent to G. Thus, $\mathbf{Hyb}_{13}$ and $\mathbf{Hyb}_{12}$ have the same output distribution.

$\mathbf{Hyb}_{14}$: In this hybrid, Sim does not follow the protocol to evaluate the circuit and sends the garbled output $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{R_i=G}$ to G on behalf of E. Instead, Sim extracts the output of $\mathcal{F}_{\mathsf{input}}^{G}$ from the commitments generated by G and reconstructs the inputs of G together with $\mathsf{mask}, \boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$. Sim then sends the inputs of G to $\mathcal{F}$ and receives G's output. Then, Sim uses $\mathsf{mask}, \boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$ and the circuit outputs of G to compute G's outputs of $\overline{C}$, i.e., $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$. Sim then computes the corresponding $\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)} = \boldsymbol{r}_{\mathbf{0}}^{(\alpha)} + (\boldsymbol{r}_{\mathbf{1}}^{(\alpha)} - \boldsymbol{r}_{\mathbf{0}}^{(\alpha)}) * \boldsymbol{s}_i$ for each $\alpha = 1, \ldots, \log 40T$, where $\boldsymbol{r}_{\mathbf{0}}^{(\alpha)}, \boldsymbol{r}_{\mathbf{1}}^{(\alpha)}$ are reconstructed from the sharings $[\boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\mathbf{1}}^{(\alpha)} - \boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]$ received by $\mathcal{O}$. In addition, E does not compute his output by himself. Instead, he directly gets his outputs from $\mathcal{F}$. Note that the shares of the input sharings generated by E for the $\geq n-t$ servers taken in $\mathbf{Hyb}_{14}$ and their corresponding wire labels are not used for the simulation. Sim does not generate them in this hybrid.

Since the garbling and evaluation in $\mathbf{Hyb}_{13}$ completely follow the protocol, from the correctness of the garbling scheme and the 2-round OT protocol, the outputs of E computed in the two hybrids are the same. If G follows the protocol to compute his output, he will also get the correct output. Note that in the explicit protocol of Π_0, when $\mathsf{mask}, \boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}$ are fixed, the computation from $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ to G's output is deterministic and invertible. Furthermore, when $\boldsymbol{r}_{\mathbf{0}}^{(\alpha)}, \boldsymbol{r}_{\mathbf{1}}^{(\alpha)}$ are fixed by the commitments, $\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}$ is also determined by $\boldsymbol{s}_i$. Therefore, we only change the order of generating G's output and the messages $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ sent to G without changing their distributions. Thus, $\mathbf{Hyb}_{14}$ and $\mathbf{Hyb}_{13}$ have the same output distribution.

Note that $\mathbf{Hyb}_{14}$ is the ideal-world scenario, where the computational advantage between any two hybrids $\mathbf{Hyb}_i$ and $\mathbf{Hyb}_{i+1}$ $(i = 0, \ldots, 13)$ is no more than $\mathsf{poly}(\kappa)/2^{\kappa}$. Thus, the total computational advantage of the adversary between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_{14}$ is still at most $\Theta(1)/2^{\kappa}$, so Π securely realizes $\mathcal{F}$ with computational security when the garbler G is corrupted.

Security against a Corrupted Evaluator. When E is corrupted, we construct the ideal adversary Sim below. We separate the description of Sim into the simulators $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ for the 3 rounds, respectively.

Simulator Sim_1

1. Sim runs the simulator $\mathsf{Sim}_{S,1}$ of the OT protocols to generate the common reference strings.
2. Sim follows the protocol to do the initialization and get the protocol Π_0'. Then, Sim sets $\mathcal{C} = \emptyset$, $\mathsf{ShCheck} = 0$, $\mathsf{CompCheck} = 0$, and $\mathsf{Comp}_j = \mathsf{Corr}_j = 0$ for each $j = 1, \ldots, n$.
3. Sim runs Steps 3-4 and Steps 6-8 of $\mathsf{Sim}_{\mathsf{pri}}$ (similar to $\mathsf{Sim}_{\mathsf{obl}}$) and obtains the garbled circuit $\tilde{C}$. In particular, Sim does not generate d_{mac}. Sim still honestly emulates $\mathcal{O}$.
4. From this step, once $\mathcal{A}$ queries a random oracle with query q:
   - If $q \notin Q_j$ for all $j \in \{1, \ldots, n\}$, Sim does nothing.
   - If q is in some Q_j, then Sim sets $\mathsf{Corr}_j = 1$ and adds S_j into $\mathcal{C}$. If $|\mathcal{C}| > 2t/3$ with $\mathsf{CompCheck} = \mathsf{ShCheck} = 0$, Sim aborts the simulation.

   Whenever Sim samples a random oracle output, if the first $\kappa + 2\log T$ bits of the random oracle outputs for two different queries, Sim aborts the simulation.
5. Sim receives $\{\tau_j\}_{j=1}^{n}$ from E and then extracts $\{\mathsf{Input}_j, \boldsymbol{r}_{\mathsf{comot}}^{S_j}\}_{j=1}^{n}$ from the RO queries it received. If such committed inputs and randomness are not found from the RO queries for some commitment, Sim sets the committed values corresponding to this commitment to $\perp$.

6. Sim receives $[\boldsymbol{v}]_{\times\kappa'}$, $\mathcal{U}$, and $\{\mathsf{Input}_j, \boldsymbol{r}_{\mathsf{comot}}^{S_j}\}_{S_j\in\mathcal{U}}$ from E.
7. Sim checks whether each S_j's share of $[\boldsymbol{v}]_{\times\kappa'}$ is computed honestly from S_j's shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$ (extracted from E's commitments). Sim also checks whether the first-round OT messages of the OT instances for S_j's input shares are computed honestly from the committed shares and the common reference string. If not, Sim sets $\mathsf{Comp}_j = 1$ and includes S_j into $\mathcal{C}$. Then:
   - If $|\mathcal{C}| \leq 2t/3$, then Sim runs Sim_0 to compute S_j's view based on the views other parties in $\mathcal{C}$. For each $i = 1, \ldots, \mathsf{rec} + n_o$, Sim samples S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for the i-th reconstruction based on the shares for servers in $\mathcal{C}$ and computes S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ accordingly by:
$$[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)} = [\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)} \oplus [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}] \otimes [\boldsymbol{s}_i]^{(2)}.$$
   - If $|\mathcal{C}| > 2t/3$ and there are only $\leq t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim aborts the simulation.

   If there are $> t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim sets $\mathsf{CompCheck} = 1$.
8. If $\mathsf{CompCheck} = 0$, for all j with $\mathsf{Comp}_j = 0$, Sim checks whether these S_j' shares of the sharings $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$ are all from valid Σ-sharings.
   - If not, Sim sets $\mathsf{ShCheck} = 1$.
   - Otherwise, Sim reconstructs the secrets, i.e., E's inputs to C, from these sharings. Sim then sends E's inputs to $\mathcal{F}$ and learns E's outputs of C, which is equal to E's output of $\overline{C}$.

Figure 44: The simulator for the first round when E is corrupted.

Simulator Sim_2

1. Sim follows the protocol to perform the check on the evaluator's input sharings. If the check fails, Sim aborts the protocol on behalf of G.
2. If $\mathsf{ShCheck} = 1$ or $\mathsf{CompCheck} = 1$ but the check passes, Sim aborts the simulation.
3. Sim generates n random strings as $c_1, \ldots, c_n$ and $n(n-1)$ random sharings as $\sigma_{i,j}$ for all pairs of servers (S_i, S_j), where the length of the strings matches the output length of $\mathcal{O}$.
4. Sim generates a random $\Sigma_{\times\kappa'}$-sharing, a random $\Sigma^{(2)}_{\times\kappa'}$-sharing, and a random $\Sigma^{(3)}_{\times\kappa'}$-sharing as the three sharings in VerSh.
5. For each $S_j \in \mathcal{C}$, Sim samples all the wire masks for $\mathtt{Circ}^{S_j}$ excluding the wire masks for output wires of XOR/output gates. Sim also samples the shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for parties in $\mathcal{C}$. Then, Sim honestly emulates $\mathcal{O}$ to compute all the outputs for queries in Q_j for all $S_j \in \mathcal{C}$, and then Sim updates the ciphertexts in $\{F_j\}_{S_j\in\mathcal{C}}$ and F_{ct} using these query outputs. In particular, for each ciphertext of the form $\mathcal{O}(q) \oplus \mathsf{m}$ for some $q \in Q_j$ with $S_j \in \mathcal{C}$, m can be obtained by honestly running the garbling process for S_j, and then the ciphertext can be computed from $\mathcal{O}(q)$ and m. Now Sim has already got GC and Versh.
6. Sim honestly samples the set $\mathcal{V}$. For each $j \in \mathcal{V}$, Sim sets $\mathsf{Corr}_j = 1$ and includes S_j into $\mathcal{C}$. Then:
   - If $|\mathcal{C}| \leq t$, then Sim runs Sim'_0 to compute S_j's view based on the views other parties in $\mathcal{C}$. In particular, E is set to be corrupted. E's output from of $\overline{C}$ is set to be his output from $\mathcal{F}$. While emulating $\mathcal{F}^E_{\mathsf{input}}$, the outputs are set to be the input sharings received from $\mathcal{F}_{\mathsf{OT}}$ after error correction. For each $i = 1, \ldots, \mathsf{rec} + n_o$, Sim samples S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for the i-th reconstruction based on the shares for servers in $\mathcal{C}$ and computes S_j's shares of $\{[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ accordingly by:
$$[\boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}]^{(3)} = [\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)} \oplus [\boldsymbol{r}_{\boldsymbol{1}}^{(\alpha)} - \boldsymbol{r}_{\boldsymbol{0}}^{(\alpha)}] \otimes [\boldsymbol{s}_i]^{(2)}.$$
   - If $|\mathcal{C}| > t$, Sim aborts the simulation.

   From this step, Sim follows Step 9 of $\mathsf{Sim}_{\mathsf{pri}}$ to emulate $\mathcal{O}$. More concretely, once $\mathcal{A}$ queries a random oracle with query q:
   - If $q \notin Q_j$ for all $j \in \{1, \ldots, n\}$, $\mathsf{Sim}_{\mathsf{pri}}$ honestly emulates the random oracle.
   - If q is in some Q_j, then if $S_j \notin \mathcal{C}$, $\mathsf{Sim}_{\mathsf{pri}}$ sets $\mathsf{Corr}_j = 1$ and adds S_j into $\mathcal{C}$. Then:

* If $|\mathcal{C}| > t$, $\mathsf{Sim}_{\mathsf{pri}}$ aborts the simulation.
* Otherwise, $\mathsf{Sim}_{\mathsf{pri}}$ runs Sim'_0 to compute S_j's view suring Π'_0 based on the views other parties in $\mathcal{C}$. Then, for each $i = 1, \ldots, \mathsf{rec} + n_o$, $\mathsf{Sim}_{\mathsf{pri}}$ samples S_j's shares of $\{[\boldsymbol{r}_{\mathbf{1}}^{(\alpha)} - \boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$ for the i-th reconstruction based on the shares for servers in $\mathcal{C}$ and computes S_j's shares of $\{[\boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ accordingly by:
$$[\boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]^{(3)} = [\boldsymbol{r}_{\boldsymbol{s}_i}^{(\alpha)}]^{(3)} \oplus [\boldsymbol{r}_{\mathbf{1}}^{(\alpha)} - \boldsymbol{r}_{\mathbf{0}}^{(\alpha)}] \otimes [\boldsymbol{s}_i]^{(2)}.$$

Then the response is determined via the following process:

* If the queried oracle is $\mathcal{O}_1$, the response to q is determined by the equation for computing a ciphertext, which is of the form
$$\mathsf{ct} = \mathcal{O}_1(q) \oplus \mathsf{m}$$
for a message m that can be computed from S_j's view and the sampled wire labels. Note that all the ciphertexts have already been sampled. Then, the response can be computed by $\mathsf{ct} \oplus m$. The remaining bits of the output from $\mathcal{O}$ for the same query are then randomly sampled. Then the output of $\mathcal{O}$ for this query is fixed.
* If the queried oracle is $\mathcal{O}_2$, the response to q is also determined by the equation for computing a ciphertext, which is of the form
$$\mathsf{ct} = \mathcal{O}_2(q) \oplus \mathsf{m}$$
for a message m that can be computed from S_j's shares of $\{[\boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]^{(3)}, [\boldsymbol{r}_{\mathbf{1}}^{(\alpha)} - \boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]\}_{\alpha=1}^{\log 40T}$. Note that all the ciphertexts have already been sampled. Then, the response can also be computed by $\mathsf{ct} \oplus m$. The remaining bits of the output from $\mathcal{O}$ for the same query are then randomly sampled. Then the output of $\mathcal{O}$ for this query is fixed.

7. For each server $S_j \in \mathcal{C}$, Sim follows the protocol to compute the corresponding second-round OT messages. For each server $S_j \notin \mathcal{C}$, Sim runs the simulator $\mathsf{Sim}_{S,2}$ to generate the second-round OT messages. Then, Sim sets the RO output of $\mathcal{O}(\mathsf{GC}\|\mathsf{Versh})$ to $\mathcal{V}\|\mathsf{randbits}$, where $\mathsf{randbits}$ is a randomly sampled string that matches the output length of $\mathcal{O}$. If $\mathsf{GC}\|\mathsf{Versh}$ has been queried by $\mathcal{A}$, Sim aborts the simulation.
8. For each $S_j \in \mathcal{C}$, Sim computes S_j's shares of $[\boldsymbol{r}_1]_{\times\kappa'}, [\boldsymbol{r}_2]^{(2)}_{\times\kappa'}, [\boldsymbol{r}_3]^{(3)}_{\times\kappa'}, [\boldsymbol{o}]^{(2)}_{\times\kappa'}$ from:
   - The four sharings in VerSh;
   - The coefficients $\{\alpha_i\}_{i=1}^{k_1}, \{\beta_i\}_{i=1}^{k_2}, \{\gamma_i\}_{i=1}^{k_3}, \{\eta_i\}_{i=1}^{k_4}$ obtained from $\mathcal{O}(\mathsf{GC})$;
   - S_j's shares of $[\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)}, \ldots, [\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)}, \ldots, [\boldsymbol{z}_{k_3}]^{(3)}$, and $[\boldsymbol{o}_1]^{(2)}, \ldots, [\boldsymbol{o}_{k_4}]^{(2)}$ obtained from the view of S_j during the execution of Π'_0 and S_j's shares of $\{[\boldsymbol{r}_{\mathbf{1}}^{(\alpha)} - \boldsymbol{r}_{\mathbf{0}}^{(\alpha)}], [\boldsymbol{r}_{\mathbf{0}}^{(\alpha)}]^{(3)}\}_{\alpha=1}^{\log 40T}$ that has been generated.

   Now, Sim has obtained $\{\mathsf{View}^{S_j}, \{\mathsf{Sh}_{i,j}, \mathsf{Sh}_{j,i}\}_{i\in\{1,\ldots,n\}\setminus\{j\}}\}_{j\in\mathcal{V}}$.
9. For each $j \in \mathcal{V}$, Sim samples a random $(\kappa + \log T)$-bit string as $\boldsymbol{r}_{\mathsf{commit}}^{S_j}$. If $\mathsf{View}^{S_j}\|\boldsymbol{r}_{\mathsf{commit}}^{S_j}\|j$ has been queried by $\mathcal{A}$, then Sim aborts the simulation. Otherwise, Sim sets the output of $\mathcal{O}(\mathsf{View}^{S_j}\|\boldsymbol{r}_{\mathsf{commit}}^{S_j}\|j)$ to c_j. Similarly, for each $j \in \mathcal{V}$ and $i > j$ or $i \notin \mathcal{V}$, Sim samples 2 random $(\kappa + \log T)$-bit strings as $\boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}, \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}$. If $\mathsf{Sh}_{i,j}\|\boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}\|i\|j$ or $\mathsf{Sh}_{j,i}\|\boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}\|j\|i$ has been queried by $\mathcal{A}$, then Sim aborts the simulation. Otherwise, Sim sets the output of $\mathcal{O}(\mathsf{Sh}_{i,j}\|\boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}\|i\|j)$ and $\mathsf{Sh}_{j,i}\|\boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}\|j\|i$ to $\sigma_{i,j}$ and $\sigma_{j,i}$ respectively. Finally, Sim sends $(\mathsf{GC}, \mathsf{VerSh}, \mathcal{V}, \{\mathsf{View}^{S_j}, \boldsymbol{r}_{\mathsf{commit}}^{S_j}\}_{j\in\mathcal{V}}, \{\mathsf{Sh}_{i,j}, \boldsymbol{r}_{\mathsf{comsh}}^{(i,j)}, \mathsf{Sh}_{j,i}, \boldsymbol{r}_{\mathsf{comsh}}^{(j,i)}\}_{i\in\mathcal{V}, j\in\{1,\ldots,n\}\setminus\{i\}})$ to E on behalf of G.

Figure 45: The simulator for the second round when E is corrupted.

Simulator Sim_3

1. Sim receives $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ from E on behalf of G.
2. Sim checks whether $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ matches what Sim generates. If not, Sim aborts the protocol on behalf of G.
3. Sim outputs what $\mathcal{A}$ outputs.

Figure 46: The simulator for the third round when E is corrupted.

We construct the following hybrids:

$\mathbf{Hyb}_0$: In this hybrid, Sim gets G's inputs and runs the protocol honestly. This corresponds to the real-world scenario.

$\mathbf{Hyb}_1$: In this hybrid, Sim additionally sets $\mathcal{C} = \emptyset$, $\mathsf{ShCheck} = 0$, $\mathsf{CompCheck} = 0$, and $\mathsf{Comp}_j = \mathsf{Corr}_j = 0$ for each $j = 1, \ldots, n$ at the beginning of the simulation. This does not affect the output distribution. Thus, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ have the same output distribution.

$\mathbf{Hyb}_2$: In this hybrid, while emulating the random oracle $\mathcal{O}$, Sim aborts the simulation if the first $\kappa + 2\log T$ bits of the RO outputs are the same for two different queries. Then, for each commitment generated by $\mathcal{O}$ from G, Sim can extract the committed inputs and randomness from the queries made by G. If such a query that matches the commitment is not found, Sim sets the committed inputs and randomness to $\perp$. Note that the RO output for each query is sampled randomly, so the probability that the first $\kappa + 2\log T$ bits of two outputs are the same is $1/2^{-(\kappa+2\log T)}$. Taking the union bound of all $< T^2$ possible pairs of queries, the probability is still less than $2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are computationally indistinguishable.

$\mathbf{Hyb}_3$: In this hybrid, Sim follows $\mathsf{Sim}_{\mathsf{pri}}$ to generate the sets $Q_1, \ldots, Q_n$ at the beginning, once $\mathcal{A}$ queries a random oracle with query q:

- If $q \notin Q_j$ for all $j \in \{1, \ldots, n\}$, Sim does nothing.
- If q is in some Q_j, then Sim sets $\mathsf{Corr}_j = 1$ and adds S_j into $\mathcal{C}$. If $|\mathcal{C}| > 2t/3$ with $\mathsf{CompCheck} = \mathsf{ShCheck} = 0$, Sim aborts the simulation.

This only changes the output distribution when $\mathcal{A}$ queries a query in $> 2t/3 > \kappa$ sets among $Q_1, \ldots, Q_n$. As argued in the proof of privacy for our garbling scheme, this only happens with $< 2^{-\kappa}$ probability. Thus, the distributions of $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ are computationally indistinguishable.

$\mathbf{Hyb}_4$: In this hybrid, Sim does not follow the protocol to generate the wire labels for the sharings $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}]$. Instead, for each wire w whose wire value is from S_j's shares of these sharings, Sim first samples wire label $k_{w,\lambda_w \oplus b_w} \| (\lambda_w \oplus b_w) \in \mathbb{F}_2^{\log 40T+1}$ randomly, where b_w is the wire value extracted from the commitments received from E. Then, Sim computes the other label by $k_{w,\lambda_w \oplus b_w \oplus} = k_{w,\lambda_w \oplus b_w} \oplus \Delta^{S_j}$ and computes the wire masks by $\lambda_w = (\lambda_w \oplus b_w) \oplus b_w$. Since $k_{w,0}$ and λ_w are uniformly randomly distributed, so is $k_{w,\lambda_w \oplus b_w} \| (\lambda_w \oplus b_w)$. We only change the order of sampling:

- λ_w and $\lambda_w \oplus b_w$;
- $k_{w,0}$ and $k_{w,1}$ when $\lambda_w \oplus b_w = 1$

without changing their distributions. Thus, $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ have the same output distribution.

$\mathbf{Hyb}_5$: In this hybrid, after receiving $[\boldsymbol{v}]_{\times\kappa'}, \mathcal{U}$ from E, Sim additionally checks whether each S_j's share of $[\boldsymbol{v}]_{\times\kappa'}$ is computed honestly from S_j's shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$ (extracted from the commitments received from E). If not, Sim sets $\mathsf{Comp}_j = 1$ and includes S_j into $\mathcal{C}$. If there are $\geq t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim sets $\mathsf{CompCheck} = 1$. This doesn't affect the output distribution. Thus, $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ have the same output distribution.

$\mathbf{Hyb}_6$: In this hybrid, after the check added in the last hybrid, if $|\mathcal{C}| > 2t/3$ and there are only $\leq t/3$ servers S_j with $\mathsf{Comp}_j = 1$, Sim aborts the simulation. Note that a server S_j with $\mathsf{Comp}_j \neq 1$ is added into $\mathcal{C}$ only because a query in Q_j has been queried. Note that until this step, Sim hasn't used the queries in Q_j to generate any messages sent to E before one of them is queried if $\mathsf{Comp}_j = 0$, which is the same case as in $\mathbf{Hyb}_2$. For the same reason, this happens with $< 2^{-\kappa}$ probability. Thus, the distributions of $\mathbf{Hyb}_6$ and $\mathbf{Hyb}_5$ are computationally indistinguishable.

$\mathbf{Hyb}_7$: In this hybrid, after the check on the servers' computation, if $\mathsf{CompCheck} = 0$, for each j with $\mathsf{Comp}_j = 0$, Sim additionally checks whether these servers S_j's shares of these sharings (extracted from the commitments received from E) are all from valid Σ-sharings. If not, Sim sets $\mathsf{ShCheck} = 1$. This does not affect the output distribution. Thus, $\mathbf{Hyb}_7$ and $\mathbf{Hyb}_6$ have the same output distribution.

$\mathbf{Hyb}_8$: In this hybrid, if the verification of E passes with $\mathsf{ShCheck} = 1$, Sim aborts the simulation. This only changes the distribution when there exists a set of $\leq t/3$ servers where each server S_j in them satisfies

$\mathsf{Comp}_j = 1$, such that the other parties' shares of an input Σ-sharing committed by E in the first round are not from a valid sharing, but they pass the check for the sharings. Note that $\geq n - t/3$ servers' shares of the sharing $[\boldsymbol{v}]_{\times\kappa'}$ must be computed from the committed shares, as $\mathsf{Comp} = 0$ for $\geq n - t/3$ parties. Assume the $\geq n - t/3$ servers' shares of a Σ-sharing $[\boldsymbol{I}_i]$ are not from a valid Σ-sharing, there is only one possible $\alpha_i \in \mathbb{F}_{2^{\kappa'}}$ that can make the sharings in VerSh valid when other coefficients are fixed. Otherwise, if there is an $\alpha_i' \neq \alpha_i$ that makes the $\geq n - t/3$ servers' shares of the sharing $[\boldsymbol{v}]_{\times\kappa'}$ valid, then the $n - t/3$ servers' shares of $(\alpha_i - \alpha_i') \cdot [\boldsymbol{I}_i]$ must comes from a valid Σ-sharing, which conflicts with the fact that the $\geq n - t/3$ servers' shares of $[\boldsymbol{I}_i]$ are invalid. Therefore, if all coefficients are uniformly randomly sampled after the servers' views are committed, the output distribution only changes with probability $\leq 2^{-\kappa'}$ when the set of $n - t/3$ parties is fixed. Taking the union bound of $< 2^n$ all such sets of servers, the probability is still $\leq 2^{-(\kappa' - n)} = 2^{-\kappa}$.

Recall that the coefficients are determined by $\mathcal{O}(s)$, where s is sampled randomly after the servers' views are committed unless $\mathcal{A}$ queries the random oracle with s before $\{\tau_j\}_{j=1}^n$ are sent to $\mathcal{A}$. When $\mathcal{A}$ never queries s before the servers' views are committed, the coefficients are uniformly random to him, and the distribution only changes with probability $\leq 2^{-\kappa}$. On the other hand, $\{\tau_j\}_{j=1}^n$ are randomly sampled RO outputs, and s contains them. Therefore, the probability that each query made by $\mathcal{A}$ to the random oracle before $\{\tau_j\}_{j=1}^n$ are sampled and sent to $\mathcal{A}$ equals s is less than $1/2^{\kappa + \log T}$. Taking the union bound of all T possible queries, the probability is still less than $1/2^{\kappa}$, so the output distribution changes with $\leq 2^{-\kappa}$ in both cases. Thus, the distributions of $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ are computationally indistinguishable.

$\mathbf{Hyb}_9$: In this hybrid, after checking the evaluator's input sharings, Sim aborts the simulation if the check passes with $\mathsf{CompCheck} = 1$. Recall that $\mathsf{CompCheck} = 1$ only when for more than $t/3$ servers S_j with $\mathsf{Corr}_j = 0$, either of the two cases happen:

- S_j's shares of $[\boldsymbol{v}]_{\times\kappa'}$ are not computed honestly from their committed shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$.
- The first-round OT message of an OT instance for S_j's input shares is not computed honestly from the committed share and the preprocessing data sent to E.

In both cases, if S_j is included in $\mathcal{U}$, the malicious behavior will be detected by G and will cause G to abort, so we do not change the output distribution.

Therefore, if $\mathcal{U}$ is uniformly random and sampled after $[\boldsymbol{v}]_{\times\kappa'}$ is sent, then the probability that κ servers S_j among the $t/3$ servers (whose shares of $[\boldsymbol{v}]_{\times\kappa'}$ are not computed honestly from their shares of $[\boldsymbol{I}_1], \ldots, [\boldsymbol{I}_{k_0}], [\boldsymbol{r}_0]_{\times\kappa'}$) are not included in $\mathcal{U}$ can be estimated by the Chernoff bound. More concretely, we add the servers into $\mathcal{U}$ one by one. For each choice of server in $\mathcal{U}$, if the current number of servers $S_j \in \mathcal{U}$ with $\mathsf{Comp}_j = 1$ is smaller than κ, then the probability that this choice of server is among these servers is at least $(t/3 - \kappa)/n$. Note that the probability that κ servers S_j among the $t/3$ servers are chosen in $\mathcal{U}$ is smaller than the probability if each time a random server (allowing repeated choices) is added into $\mathcal{U}$. In this case, each time a server is added, let i be the number of servers in $\mathcal{U}$ after adding this server, then we define:

- $X_i = 1$ if the chosen party is not among a randomly chosen set of $t/3 - \kappa$ servers that are not included in $\mathcal{U}$ before.
- $X_i = 0$ otherwise.

Then, $X_1, \ldots, X_{t/3}$ are independent, with the probability of each $X_i = 1$ be $(t/3 - \kappa)/n$. Let $X = X_1 + \cdots + X_{t/3}$. Then, $\mu := \mathbb{E}(X) = t(t/3 - \kappa)/3n > 4(2\kappa + \log T)$, so by Chernoff Bound:

$$P(X \leq \kappa) \leq P(X \leq (1 - 3/4)\mu) \leq e^{-\frac{9}{32} \cdot \mu} \leq 2^{-\kappa} \cdot 2^{-\kappa} \cdot T^{-1}.$$

Thus, if $\mathcal{U}$ is uniformly random, the output distribution only changes with probability $1 - (1 - 2^{-\kappa} \cdot 2^{-\kappa} \cdot T^{-1}) \cdot (1 - T^{-\kappa}) < 2^{-\kappa} \cdot T^{-1}$.

Note that when $[\boldsymbol{v}]_{\times\kappa'}$ is not queried by $\mathcal{A}$ to $\mathcal{O}$, then $\mathcal{U}$ is indeed uniformly random. In this case, the distribution only changes with no more than $2^{-\kappa} \cdot T^{-1}$ probability. On the other hand, $\mathcal{A}$ has at most T

queries to $\mathcal{O}$, where each time $\mathcal{A}$ needs to fix $[\boldsymbol{v}]_{\times\kappa'}$ and learn whether it can pass the check, so each time $[\boldsymbol{v}]_{\times\kappa'}$ only passes the check with no more than $2^{-\kappa}\cdot T^{-1}$ probability. Taking the union bound of T queries, the probability that there exists a query of the form $[\boldsymbol{v}]_{\times\kappa'}$ by $\mathcal{A}$ that can pass the check (with more than $t/3$ servers S_j' shares of $[\boldsymbol{v}]_{\times\kappa'}$ are not computed honestly from their shares of $[\boldsymbol{I}_1],\ldots,[\boldsymbol{I}_{k_0}],[\boldsymbol{r}_0]_{\times\kappa'}$ with $\mathsf{Corr}_j=0$) is no more than $T\cdot 2^{-\kappa}\cdot T^{-1}=2^{-\kappa}$. If it does not exist, $\mathcal{A}$ either sends a query he made before, which would not change the distribution, or he will send another sharing $[\boldsymbol{v}]_{\times\kappa'}$ that he has not queried, which changes the output distribution with no more than $2^{-\kappa}$ probability. Therefore, the total probability that the output distribution changes is no more than $2^{-\kappa}+(1-2^{-\kappa})\cdot 2^{-\kappa}=\Theta(1)/2^{\kappa}$. Thus, the distributions of $\mathbf{Hyb}_9$ and $\mathbf{Hyb}_8$ are computationally indistinguishable.

$\mathbf{Hyb}_{10}$: In this hybrid, Sim does not follow the protocol to compute $c_1,\ldots,c_n$ and $\sigma_{i,j}$ for each pair of servers (S_i,S_j). Instead, Sim directly samples random strings (with the length as an RO output) as them. Then, the corresponding RO outputs are set to these values.

Note that if $\mathcal{A}$ has not queried $\mathcal{O}$ before they are sent, we do not make any changes. Therefore, the output distribution only changes when one of the queries has been made by $\mathcal{A}$. However, each of the queries contains a uniformly random $(\kappa+\log T)$-bit string, and it is not used to compute any message sent to $\mathcal{A}$ before sending the commitments. Therefore, the probability that the query has been made is no more than $T/2^{\kappa+\log T}=2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_{10}$ and $\mathbf{Hyb}_9$ are computationally indistinguishable.

$\mathbf{Hyb}_{11}$: In this hybrid, Sim does not follow the protocol to sample $[\boldsymbol{r}_1]_{\times\kappa'},[\boldsymbol{r}_2]^{(2)}_{\times\kappa'},[\boldsymbol{r}_3]^{(3)}_{\times\kappa'},[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ and compute the 4 sharings in VerSh. Instead, Sim samples the 4 sharings in VerSh randomly and then computes $[\boldsymbol{r}_1]_{\times\kappa'},[\boldsymbol{r}_2]^{(2)}_{\times\kappa'},[\boldsymbol{r}_3]^{(3)}_{\times\kappa'},[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ based on VerSh, the coefficients computed from $\mathcal{O}(\mathsf{GC})$, $[\boldsymbol{x}_1],\ldots,[\boldsymbol{x}_{k_1}]$, $[\boldsymbol{y}_1]^{(2)},\ldots,[\boldsymbol{y}_{k_2}]^{(2)}$, $[\boldsymbol{z}_1]^{(3)},\ldots,[\boldsymbol{z}_{k_3}]^{(3)}$, and $[\boldsymbol{o}_1]^{(2)},\ldots,[\boldsymbol{o}_{k_4}]^{(2)}$. Since $[\boldsymbol{r}_1]_{\times\kappa'},[\boldsymbol{r}_2]^{(2)}_{\times\kappa'},[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$ are uniformly random $\Sigma_{\times\kappa'},\Sigma^{(2)}_{\times\kappa'},\Sigma^{(3)}_{\times\kappa'}$-sharings, and $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ is a uniformly random $\Sigma^{(2)}$-sharing with an all-zero secret, so are the corresponding sharings in VerSh. Therefore, we only change the order of them without changing their distributions. Thus, $\mathbf{Hyb}_{11}$ and $\mathbf{Hyb}_{10}$ have the same output distribution.

$\mathbf{Hyb}_{12}$: In this hybrid, Sim does not follow the protocol to generate the garbled circuit. Instead, for each server S_j that is not in the set $\mathcal{C}$ after performing the verification and sampling VerSh, Sim does not follow the protocol to query the random oracle with the queries in Q_j first and then compute the ciphertexts in $\tilde{C}$ with these outputs. Instead, Sim follows $\mathsf{Sim}_{\mathsf{obl}}$ to generate these ciphertexts and emulate $\mathcal{O}$ in the future simulation. Following from the proof of privacy of our garbling scheme, the generated $\tilde{C}$ has the same distribution as computed by following the protocol, and the emulation of $\mathcal{O}$ can be programmed when $\mathcal{A}$ queries an element in Q_j for no more than t of $j\in\{1,\ldots,n\}$. Note that the queries in Q_j are not used for sending any message to $\mathcal{A}$ if $S_j\notin\mathcal{V}$ and $\mathsf{Comp}_j=0$. The distribution only changes when $\mathcal{A}$ queries $t/3$ of them. For the same reason as in $\mathbf{Hyb}_2$, this happens with at most $2^{-\kappa}$ probability. Thus, the distributions of $\mathbf{Hyb}_{12}$ and $\mathbf{Hyb}_{11}$ are computationally indistinguishable.

Note that for the servers that are never included in $\mathcal{C}$, Sim does not need their shares of $[\boldsymbol{r}_1]_{\times\kappa'}$, $[\boldsymbol{r}_2]^{(2)}_{\times\kappa'}$, $[\boldsymbol{r}_3]^{(3)}_{\times\kappa'}$, and $[\boldsymbol{o}]^{(2)}_{\times\kappa'}$ in the simulation. Sim does not generate them in the future hybrids. In addition, Sim does not need the set $\{\boldsymbol{r}^{(1)}_{\boldsymbol{s}_i\oplus 1},\ldots,\boldsymbol{r}^{(\log 40T)}_{\boldsymbol{s}_i\oplus 1}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ during the simulation, Sim also does not generate them in future hybrids.

$\mathbf{Hyb}_{13}$: In this hybrid, Sim does not sample $\boldsymbol{r}^{S_j}_{\mathsf{commit}}$ and $\boldsymbol{r}^{(i,j)}_{\mathsf{comsh}}$ if $i,j\notin\mathcal{V}$ and check whether the corresponding query to the random oracle has been queried. Note that Sim does not need these random strings except that Sim checks whether these queries are made. Therefore, this only changes the output distribution when one of them is truly queried. Since they are not used for sending any messages to $\mathcal{A}$, each query made by $\mathcal{A}$ only has no more than $1/2^{\log T+\kappa}=T^{-1}\cdot 2^{-\kappa}$ probability to match any of the queries. Taking the union bound for all T possible queries made by $\mathcal{A}$, the probability is still $\leq 2^{-\kappa}$. Thus, the distributions of $\mathbf{Hyb}_{13}$ and $\mathbf{Hyb}_{12}$ are computationally indistinguishable.

$\mathbf{Hyb}_{14}$: In this hybrid, during the simulation of Round 2, for the OT instances corresponding to input shares of servers out of $\mathcal{C}$, Sim does not follow the protocol to compute OT_2 and generate the second-round OT messages. Sim also changes his way of generating the common reference strings. More concretely, Sim runs the simulator $\mathsf{Sim}_{S,1}$ for the OT protocols to generate all the common reference strings and runs $\mathsf{Sim}_{S,2}$ for the OT protocols to generate the second-round OT messages for the OT instances of servers out of $\mathcal{C}$.

For those input shares of servers in $\mathcal{C}$, the common reference strings from K_{OT} and from $\mathsf{Sim}_{S,1}$ should be indistinguishable to $\mathcal{A}$. If not, as long as E honestly runs the OT protocol, there cannot exist a $\mathsf{Sim}_{S,2}$ to make the joint distribution of simulated σ, ots_2 indistinguishable from the real-world messages to $\mathcal{A}$, which breaks the sender's security. Moreover, for the OT instances corresponding to input shares of servers out of $\mathcal{C}$, we only change the way of generating σ, ots_2. From the sender's security, the joint distributions of σ, ots_2 from the two different ways are indistinguishable to the adversary $\mathcal{A}$. We argue that the distributions of $\mathbf{Hyb}_{14}$ and $\mathbf{Hyb}_{13}$ are computationally indistinguishable by constructing hybrids $\mathbf{Hyb}_{14.1}, \ldots, \mathbf{Hyb}_{14.N_{\mathsf{OT}}}$, where N_{OT} is the number of executions of the OT protocol.

$\mathbf{Hyb}_{14.i}$: In this hybrid, Sim does not follow the protocol to compute the common reference string and G's second-round OT messages for the i-th OT instance corresponding to S_j's input shares for $S_j \notin \mathcal{C}$. Instead, Sim runs the simulator $\mathsf{Sim}_{S,1}$ for the OT protocol to generate the common reference string in these OT instances and runs $\mathsf{Sim}_{S,2}$ to generate the second-round OT message. Let $\mathbf{Hyb}_{14.0} = \mathbf{Hyb}_1 3$, we argue that the distributions of $\mathbf{Hyb}_{14.i}$ and $\mathbf{Hyb}_{14.i-1}$ are computationally indistinguishable.

For the sake of contradiction, assume that $\mathbf{Hyb}_{14.i}$ and $\mathbf{Hyb}_{14.i-1}$ are distinguishable, so there exists an adversary $\mathcal{A}$ in our model that can distinguish the output distributions before and after we change σ, ots_2 to the simulated messages. Now we construct an adversary $\mathcal{A}'$ in our model that breaks the sender's security of the i-th OT. Concretely, $\mathcal{A}'$, with the auxiliary input of all the common reference strings of other OT protocols, receives a string from the challenger, which is σ, ots_2 either computed from the real protocols or the simulators. Then $\mathcal{A}'$ runs Sim with the change that σ, ots_2 are replaced by the string received from the challenger. Finally, $\mathcal{A}'$ uses $\mathcal{A}$ to distinguish whether the challenge is computed from the real protocols or the simulators. Note that the advantage of $\mathcal{A}'$ is the same as that of $\mathcal{A}$, which breaks the security of the i-th OT protocol. This leads to a contradiction.

Thus, $\mathbf{Hyb}_{14.i}$ and $\mathbf{Hyb}_{14.i-1}$ have the same output distribution.

Note that $\mathbf{Hyb}_{14.N_{\mathsf{OT}}}$ is exactly $\mathbf{Hyb}_{14}$, so the distributions of $\mathbf{Hyb}_{14}$ and $\mathbf{Hyb}_{13}$ are computationally indistinguishable.

$\mathbf{Hyb}_{15}$: In this hybrid, Sim does not follow the protocol to check the garbled outputs received from the evaluator. Instead, Sim directly checks whether $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ matches what Sim generates. As analyzed in the proof of authenticity of our garbling scheme, this only changes the output distribution when E outputs $Y = (\kappa, \{\boldsymbol{s}_i', \boldsymbol{r}_{\boldsymbol{s}_i'}^{(1)} \ldots, \boldsymbol{r}_{\boldsymbol{s}_i'}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o})$ such that the check passes, but $\{\boldsymbol{s}_i'\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ flips $t+1$ bits from $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$. Since Sim does not generate $\{\boldsymbol{r}_{\boldsymbol{s}_i \oplus \mathbf{1}}^{(\alpha)}\}_{\alpha=1}^{\log 40T}$ for $i = \mathsf{rec}+1, \ldots, \mathsf{rec}+n_o$ during the simulation, an adversary who flips $\geq t+1$ bits from $\{\boldsymbol{s}_i\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ can only passes the check with probability $< 2^{-\kappa}$ using a similar analysis as in the proof of authenticity. Thus, the distributions of $\mathbf{Hyb}_{15}$ and $\mathbf{Hyb}_{14}$ are computationally indistinguishable.

$\mathbf{Hyb}_{16}$: In this hybrid, G does not compute his output by himself. Instead, he directly gets his outputs from $\mathcal{F}$, where the evaluator's inputs to $\mathcal{F}$ are reconstructed from the input sharings using error correction. Note that the set $\{\boldsymbol{s}_i, \boldsymbol{r}_{\boldsymbol{s}_i}^{(1)}, \ldots, \boldsymbol{r}_{\boldsymbol{s}_i}^{(\log 40T)}\}_{i=\mathsf{rec}+1}^{\mathsf{rec}+n_o}$ received from E match what G generates, we can regard that E honestly run the evaluation algorithm of the garbling scheme. From the correctness of the garbling scheme, G's outputs $\overline{y}_G$ of $\overline{C}$ obtained from the protocol execution is exactly $\mathsf{En}(y_G \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}))$ for G's output y_G from $\mathcal{F}$. The protocol output of G is decoded from $\mathsf{En}(y_G \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}))$ with De. Recall that

$$\mathsf{De}(\mathsf{En}(y_G \oplus \mathsf{mask}, (\boldsymbol{r}_{\mathsf{en}}^{(1)}, \ldots, \boldsymbol{r}_{\mathsf{en}}^{(W_O/k)}))) = y_G \oplus \mathsf{mask},$$

the protocol outputs $\mathsf{De}(\overline{y}_G) \oplus \mathsf{mask}$ is the same as the functionality output y_G. Thus, $\mathbf{Hyb}_{16}$ and $\mathbf{Hyb}_{15}$ have the same output distribution.

Note that $\mathbf{Hyb}_{16}$ is the ideal-world scenario, where the statistical difference between any two hybrids $\mathbf{Hyb}_i$ and $\mathbf{Hyb}_{i+1}$ $(i = 0, \ldots, 15)$ is no more than $\mathsf{poly}(\kappa)/2^\kappa$. Thus, the total difference between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_{17}$ is still at most $\mathsf{poly}(\kappa)/2^\kappa$, so Π securely realizes $\mathcal{F}$ with computational security when the evaluator E is corrupted.