Three-Round Secure Multiparty Computation from Black-Box Two-Round Oblivious Transfer

Arpita Patra
Indian Institute of Science

Akshayaram Srinivasan
Tata Institute of Fundamental Research

Abstract

We give constructions of three-round secure multiparty computation (MPC) protocols for general functions that make *black-box* use of a two-round oblivious transfer. For the case of semi-honest adversaries, we make use of a two-round, semi-honest secure oblivious transfer in the plain model. This resolves the round-complexity of black-box (semi-honest) MPC protocols from minimal assumptions and answers an open question of Applebaum et al. (ITCS 2020). For the case of malicious adversaries, we make use of a two-round maliciously-secure oblivious transfer in the common random/reference string model that satisfies a (mild) variant of adaptive security for the receiver.

Contents

1 Introduction

Secure Multiparty Computation (MPC) is a fundamental cryptographic primitive that allows a set of mutually distrusting parties to compute a joint function of their private inputs. The security guarantee provided here is that any adversary corrupting an arbitrary subset of the participating parties cannot learn anything about the inputs of the honest parties except what is leaked from the output of the function. The seminal feasibility results of Yao [Yao86] and Goldreich, Micali, and Wigderson [GMW87] showed that any multiparty functionality can be securely computed.

An important line of research in this area aims to construct efficient MPC protocols that minimizes the *number of rounds of communication.* The work of Beaver, Micali, and Rogaway [BMR90] initiated this research direction and gave a construction of a constant-round protocol for computing general functions. On the lower bounds side, it is known that a single-round of communication is insufficient for securely computing most functionalities and hence, the minimum number of rounds needed to securely compute general functions is two.

A recent line of work has led to constructions of round-optimal (i.e., two-round) secure multiparty computation protocols under various cryptographic assumptions. The work of Garg et al. [GGHR14] gave a construction of such a protocol based on indistinguishability obfuscation [BGI⁺01, GGH⁺13] and subsequent work of Gordon et al. [GLS15] improved the assumption to a witness encryption scheme [GGSW13]. Later, Mukherjee and Wichs [MW16] (and the subsequent works [BP16, PS16]) gave a protocol based on the Learning with Errors assumption [Reg05], Garg and Srinivasan [GS17] gave a construction from Bilinear maps and Boyle et al. [BGI17, BGI⁺18] gave a construction from the Decisional Diffie-Hellman (DDH) assumption. Finally, the works of Benhamouda and Lin [BL18] and Garg and Srinivasan [GS18] gave constructions of two-round MPC protocols based on the minimal assumption that two-round oblivious transfer (OT) exists.

Black-Box Round Complexity. A cryptographic protocol P is said to make *black-box* use of an underlying primitive Q if P only makes input/output calls to Q and is agnostic to how Q is implemented. Apart from being a fundamental theoretical question, black-box protocols tend to be more efficient than their non-black-box counterparts and are usually viewed as the first step towards practicality. Unfortunately, the constructions of two-round MPC protocols from [BL18, GS18] made non-black-box use of a two-round OT. On the other hand, a recent work of Applebaum et al. [ABG⁺20] showed that such non-black-box use is inherent by providing a black-box separation between these two primitives. As far as positive results are concerned, we do know of 4-round MPC protocols making black-box use of a two-round OT from [ACJ17, GIS18, LLW20]. These works left open the following intriguing question (which was explicitly mentioned in [ABG⁺20]):

Can we construct a three-round secure multiparty computation protocol for general functions making black-box use of a two-round OT?

1.1 Our Results

In this work, we give a near complete answer to the above question. For the case of semi-honest adversaries, we fully resolve the problem and show that two-round OT is black-box complete for three-round MPC. Specifically,

Informal Theorem 1.1. *Let f be an arbitrary multiparty functionality. There exists a three-round protocol that securely computes f against semi-honest adversaries corrupting an arbitrary subset of* *the parties. The protocol makes black-box use of a two-round, semi-honest secure OT and is in the plain model. The computational cost of the protocol grows polynomially with the circuit size of f and the security parameter.*

For the case of malicious adversaries, we give a three-round MPC protocol that makes black-box use of two-round, malicious-secure OT that additionally satisfies an equivocality property for the receiver's message. Specifically, we require the existence of a special algorithm that can equivocate the first round receiver OT message to both bits 0 and 1. Such equivocality property is implied by a two-round OT that is secure against a malicious adversary that can adaptively corrupt the receiver or, it can be obtained from black-box use of a dual-mode public-key encryption scheme [PVW08]. The main theorem we show for malicious adversaries is the following:

Informal Theorem 1.2. *Let f be an arbitrary multiparty functionality. There exists a three-round protocol that UC-realizes f (with unanimous abort) against malicious adversaries corrupting an arbitrary subset of the parties. The protocol makes black-box use of a two-round, UC-secure OT against malicious adversaries with equivocal receiver security and is in the common random/reference string model. The computational cost of the protocol grows polynomially with the circuit size of f and the security parameter.*

We note that the work of Garg and Srinivasan [GS18] gave a generic transformation from any two-round, malicious-secure OT to one that additionally satisfies the equivocal receiver property. Unfortunately, this transformation makes non-black-box use of a PRG (but makes black-box use of OT). We leave open the interesting problem of obtaining a black-box transformation, or showing that such non-black-box use is inherent.

2 Technical Overview

In this section, we give a high-level overview of the main techniques used in the construction of our MPC protocols in the semi-honest and the malicious setting.

Starting Point. Our work builds on the recent results of [BL18, GS18] which gave constructions of two-round secure multiparty computation from two-round oblivious transfer. The key technical contribution in these works is the design of a round-collapsing compiler that takes a larger round protocol for securely computing the required functionality and squishes the number of rounds to two. Specifically, instead of the parties interacting with each other as in the larger round protocol, the round-collapsing compiler gave a mechanism wherein the garbled circuits generated by each party performs this interaction. The interaction between garbled circuits is enabled by making use of a two-round oblivious transfer. Unfortunately, these constructions [BL18, GS18] require non-black-box use of cryptographic primitives.

If we look closely into these constructions, we observe that there is only one place where non-black-box use of cryptography is needed. Specifically, the garbled circuits which perform the interaction on behalf of the parties use the code of the underlying larger round protocol. Thus, if the larger round protocol makes use of cryptographic primitives such as an oblivious transfer, then the squished protocol makes non-black-box use of these primitives. On the other hand, if the larger round protocol only made use of information-theoretic operations, then the resultant two-round protocol makes black-box use of cryptography. Unfortunately, the negative results in [Kus89] rules out information-theoretic secure computation protocols for most functions in the dishonest majority setting. Furthermore, the work of Applebaum et al. [ABG+20] showed that such non-black-box use of oblivious transfer is inherent if we want to construct a two-round MPC protocol. However, their work left open the problem of constructing a black-box three-round MPC protocol based on two-round oblivious transfer.

The work of Garg, Ishai, and Srinivasan [GIS18] observed that if the parties apriori shared random OT correlations, then one can use the results of [Kil88, IPS08] to construct an information-theoretic MPC protocol in the OT correlations model. Now, squishing the number of rounds of such a protocol using the round-collapsing compiler of [BL18, GS18] gives rise to an MPC protocol that makes black-box use of cryptography. Garg et al. [GIS18] also gave a method of generating such correlations in a single round using a primitive called *non-interactive oblivious transfer*. This gives rise to the following three-round protocol that makes black-box use of cryptographic operations: use the first round to generate random OT correlations relying on non-interactive oblivious transfer, and use the next two rounds to implement the round-collapsing compiler of [BL18, GS18]. However, a non-interactive oblivious transfer is a stronger primitive and it is not known whether this can be constructed from a two-round oblivious transfer.

Double Selection Functionality. If we abstract out the other details from [GIS18], then the main ingredient needed to instantiate the black-box version of the round-collapsing compiler is a three-round protocol for a special multiparty functionality that we call as the *double selection*. In this functionality, only three of the n parties, say, P_1, P_2 and P_3 have private inputs. The input of P_1 is given by two bits (α, r), the input of P_2 is given by two bits (x_0, x_1) and the input of P_3 is given by two strings (y_0, y_1). The functionality first computes $x_\alpha \oplus r$ and then computes $y_{x_\alpha \oplus r}$ and delivers $(x_\alpha \oplus r, y_{x_\alpha \oplus r})$ to every party (and not just to P_1, P_2, and P_3.). In other words, the functionality first selects x_α from (x_0, x_1), XORs x_α with r and then again selects $y_{x_\alpha \oplus r}$ from (y_0, y_1) and hence, the name double selection. The work of Garg et al. [GIS18] can be viewed as giving a three-round protocol for the double selection functionality based on non-interactive oblivious transfer. The goal of this work is to give such a protocol based only on black-box use of a two-round oblivious transfer.

We first note that if we relax the requirement to say that, only one of $\{P_1, P_2, P_3\}$ gets the output at the end of the third round, then based on prior work, it is possible to design a black-box three-round protocol for this relaxed functionality. Indeed, one can express the double selection functionality as a degree-3 polynomial (over $\mathbb{F}_2$) and use the protocol from [ACJ17] to securely evaluate a degree-3 polynomial. Additionally, it is not too hard to see that if we invoke such a protocol thrice, then we can enable each one of $\{P_1, P_2, P_3\}$ to get the output of the double selection functionality at the end of the third round. However, the main technical challenge here is to enable each of the n parties and not just $\{P_1, P_2, P_3\}$, to reconstruct the output at the end of the third round. This requirement is equivalent to constructing a three-party protocol with a special property called as *publicly-decodable transcript* [ABG+20]. Roughly speaking, this property requires the existence of an efficient algorithm that takes the transcript of the three-party protocol and gives the output of the double selection functionality. For the sake of simplicity, let us restrict ourselves to protocols where the last round (i.e., the third round) message contains the output in the clear. We now explain how to construct such a protocol making black-box use of two-round oblivious transfer.

Key Idea: "Cascading Oblivious Transfer." Since the last round message of the protocol contains the output of the functionality in the clear, this implies that there exists some party that can compute this output at the end of the second round and then broadcast this value to all the parties in the third round. This seems particularly challenging if we restrict ourselves to making black-box use of a two-round oblivious transfer. Indeed, this implies that we need a mechanism to compute the output of a degree-3 function in two rounds using a two-round oblivious transfer that only enables degree-2 computation. This apparent mismatch in the degree is the key challenge that we need to tackle.

This is where our idea of "cascading oblivious transfer" comes into the picture. Specifically, in our protocol, one of the parties, say P_3, computes a sender OT message with respect to a receiver OT message generated by P_1 (that encodes P_1's input). The sender inputs used by P_3 to generate this message are in fact, two other sender OT messages computed by P_3 with respect to a receiver OT message generated by P_2 (that encodes P_2's input). Thus, the "inner" sender OT message encodes a degree two computation of P_2 and P_3's inputs and the "outer" sender OT message encodes a degree-3 computation of P_1, P_2 and P_3's inputs. This idea of cascading two sender OT messages by P_3 allows P_1 to compute a degree-3 function in two rounds and thus, enabling us to solve the degree mismatch problem. Let us first see how to implement this "cascading oblivious transfer" idea in the semi-honest setting and later explain the additional challenges that arise in the malicious setting.

2.1 Semi-Honest Setting

In the first round, P_1 computes two receiver OT messages: otr that encodes α as the choice bit and otr' that encodes r as the choice bit. In parallel, P_2 computes two receiver OT messages otr_0 that encodes its input x_0 and otr_1 that encodes x_1. P_1 broadcasts $(\mathsf{otr}, \mathsf{otr}')$ and P_2 broadcasts $(\mathsf{otr}_0, \mathsf{otr}_1)$ in the first round. In the second round, P_3 chooses a random bit mask and computes two sender OT messages: ots_0 with respect to otr_0 using $(y_0 \oplus \mathsf{mask}, y_1 \oplus \mathsf{mask})$ as its sender inputs and ots_1 with respect to otr_1 using again $(y_0 \oplus \mathsf{mask}, y_1 \oplus \mathsf{mask})$ as its inputs. It then computes the "cascading" sender OT message ots with respect to otr using $(\mathsf{ots}_0, \mathsf{ots}_1)$ as its two sender messages. Additionally, it computes ots' with respect to otr' with $(\mathsf{mask}, y_1 \oplus y_0 \oplus \mathsf{mask})$ as its sender messages. It then sends $(\mathsf{ots}, \mathsf{ots}')$ to P_1 in the second round.

Now, the randomness used in generating otr enables P_1 to recover ots_α from ots. However, recall that ots_α is generated with respect to otr_α and the randomness used for generating this message is available with P_2. Thus, to enable P_1 to decrypt ots_α, in the second round, P_2 computes a sender OT message with respect to otr with the input and randomness used for computing otr_0 and otr_1 as the two sender inputs. Thus, P_1 can first recover x_α and the randomness used for generating otr_α from P_2's second round message and then obtain $y_{x_\alpha} \oplus \mathsf{mask} := x_\alpha(y_1 \oplus y_0) \oplus y_0 \oplus \mathsf{mask}$ from ots_α. P_1 also computes $r(y_1 \oplus y_0) \oplus \mathsf{mask}$ from ots' using the randomness used in generating otr'. It adds these two values to get $y_{x_\alpha \oplus r}$. In the last round, P_1 broadcasts $(x_\alpha \oplus r, y_{x_\alpha \oplus r})$. This protocol satisfies correctness and we can show that this protocol is secure against semi-honest adversaries by relying on the semi-honest security of the two-round oblivious transfer.

From Double Selection to General Functions. To give a protocol for general functions, we can use the reduction from general functions to double selection implicit in the work of [GIS18]. Alternatively, we can use the above idea of cascading oblivious transfer to give a three-round secure protocol for a related degree-3 function called as $\mathsf{3MULTPlus}$. We can then rely on completeness results from [BGI+18, GIS18, ABG+20] who showed a round-preserving black-box reduction from a semi-honest protocol for computing general functions to a secure protocol for 3MULTPlus functionality. In the main body, we construct a protocol for securely computing 3MULTPlus and directly rely on the above completeness theorem to give a self-contained version of our semi-honest MPC result.

2.2 Malicious Setting

In the malicious setting, many other challenges arise and we now explain our ideas to solve them.

Challenge-1: Attack by a malicious P_3. Let us start with the bare-bones version of the malicious protocol which is just the semi-honest protocol but with all the oblivious transfer invocations replaced with a malicious secure version. On inspection, we see that a corrupt P_3 can completely break the security of this protocol. Specifically, P_3 can compute ots_0 and ots_1 on two different pairs of inputs, say using $(\mathsf{mask}, \mathsf{mask})$ and $(1 \oplus \mathsf{mask}, 1 \oplus \mathsf{mask})$ respectively and compute ots' on inputs $(\mathsf{mask}, \mathsf{mask})$. Depending on the message received from P_1 in the last round, corrupt P_3 learns the value α. In order to prevent such an attack, we need a mechanism to ensure that P_3 uses consistent inputs to compute both ots_0 and ots_1.

One way to ensure consistency of P_3's inputs is to ask P_3 to give a zero-knowledge proof that the inputs used in both these computations are consistent. However, a naïve way of implementing such a zero-knowledge proof makes non-black-box use of cryptographic primitives which we want to avoid. To give a "black-box" zero-knowledge proof, we make use of "MPC-in-the-head" approach of Ishai et al. [IKOS07].

Solution: "MPC-in-the-head" Approach. To convey the main idea, we first explain a simple solution that blows-up the number of rounds and later show how to squish the number of rounds. P_3 imagines m-servers in its head (for some appropriately chosen parameter m). It then shares y_0, y_1, mask among these m servers using a threshold linear secret sharing scheme with a threshold parameter t. For each $i \in [m]$, P_3 computes $\{\mathsf{ots}_0^i, \mathsf{ots}_1^i, \mathsf{ots}^i, \mathsf{ots}'^i\}$ using the shares given to the i-th server. Specifically, the values $(y_0, y_1, \mathsf{mask})$ in the original computation are replaced with the shares $(y_0^i, y_1^i, \mathsf{mask}^i)$ given to the i-th server. P_3 sends $\{\mathsf{ots}^i, \mathsf{ots}'^i\}_{i \in [m]}$ to P_1 in the second round. P_1 now chooses a random subset T of $[m]$ of size t and asks P_3 to reveal the shares and the randomness used in the computation of $(\mathsf{ots}^i, \mathsf{ots}'^i)$ for every $i \in T$. P_1 now checks if these computations are correct. If they are all correct, then for each $i \in [m]$, P_1 recovers the share of the output and reconstructs the output. Here, we are crucially relying on the fact $x_\alpha(y_1 \oplus y_0) \oplus y_0 \oplus \mathsf{mask}$ and $r(y_1 \oplus y_0) \oplus \mathsf{mask}$ recovered by P_1 in the bare-bones protocol are linear functions of y_0, y_1, mask and the secret sharing scheme used by P_3 supports linear operations on the shares. This ensures that P_1 can recover the correct output from the shares. However, this idea seems to blow-up the number of rounds to 4. To squish the number of rounds to 2, we make use of the [JKKR17] trick, wherein P_1, in the first round, uses a t-out-of-m oblivious transfer to commit to its set T and P_3 in the second round uses the m sets of inputs, randomness as its sender inputs.

We can now show that if a malicious P_3 is using inconsistent inputs in "many" server executions then it gets caught with overwhelming probability. On the other hand, if P_3 is using inconsistent inputs in a "small" number of server executions, then we can rely on the error correcting properties of the secret sharing scheme to recover the correct output.[1]

[1] Here, we need to additionally ensure that malicious P_3 is generating the shares correctly. Hence, we make use of

Need for Equivocal Receiver Security. Here, another technical issue arises and to solve this, we need the oblivious transfer to satisfy the equivocality property on the receiver's message. To see why this additional property is required, consider the case where P_2 is honest but P_1 is corrupted. Since the adversary is rushing, the honest P_2 sends both $\mathsf{otr}_0, \mathsf{otr}_1$ before receiving $\mathsf{otr}, \mathsf{otr}'$. Recall that in the second round, P_2 generates a sender OT message with respect to otr with the input and the randomness used in otr_0 and otr_1 as its OT inputs. Unfortunately, this leads to the following issue during simulation. We cannot know the value of x_α unless we receive otr from the corrupt P_1. This value is obtained only after we send both $\mathsf{otr}_0, \mathsf{otr}_1$. However, since x_α and the randomness used in generating otr_α are needed to compute the sender OT message from P_2, we need to generate otr_α in a way that it correctly encodes x_α. To solve this issue, we rely on the equivocality property of the receiver's message. Specifically, since the first round OT message of the receiver can be equivocated to both bits 0 and 1, we use the equivocal simulator to generate randomness that is consistent with the encoding of x_α. We then use this randomness to generate the second round OT message. As mentioned earlier, this property is satisfied by any two-round oblivious transfer that is secure against adversaries that can adaptively corrupt the receiver or it can be obtained from a dual-mode public-key encryption scheme [PVW08].

Challenge-2: Attack by Malicious P_2. In the previous step, we prevented a malicious P_3 from breaking the security of the protocol. However, we observe that a malicious P_2 can still break the security of the protocol by mounting an input dependent abort. Specifically, a corrupt P_2 can generate the second round OT message with respect to otr such that only one of its two sender inputs contains the correct randomness used in generating $(\mathsf{otr}_0, \mathsf{otr}_1)$. It sets the other sender input to be some junk value. If the input α of P_1 corresponds to the position that contains the junk value, then P_1 aborts at the end of the second round. This enables P_2 to learn the value α. A first natural idea to prevent this attack is to use a zero-knowledge proof to show that P_2 is using the correct inputs in generating the sender OT message. However, unlike the previous step, the relation that we want to prove (or equivalently, the functionality computed by the MPC) involves a cryptographic statement and in those cases, the "MPC-in-the-head" approach leads to non-black-box use of cryptographic primitives. Thus, we need a new approach to deal with this issue.

Solution: Using an OT-Combiner. We first observe that if the input α of P_1 was uniformly random, then the probability that a corrupt P_2 can guess α to force P_1 to abort is $1/2$. For $\kappa = \Omega(\lambda)$ (where λ is the security parameter), consider invoking the above protocol κ times on independently chosen random P_1 inputs $(\alpha_1, \ldots, \alpha_\kappa)$. Then, the probability that corrupt P_2 can guess more than λ of these inputs is negligible. Given this observation, consider the following two-party functionality:

1. The input of P_1 is given by two bits (α, r) and the input of P_2 is given by two other bits (x_0, x_1).

2. P_1 and P_2 also share $\kappa = \Omega(\lambda)$ random OT correlations with P_1 acting as the receiver and P_2 acting as the sender. Additionally, a corrupt P_2 might learn λ of these receiver correlations. We call these as "leaky" OT correlations.

3. At the end of the protocol, we want to both P_1 and P_2 to learn $(x_\alpha \oplus r)$.

---

a pairwise verifiable secret sharing based on bivariate polynomials and do additional checks on the shares to ensure that the sharing is done correctly.

A statistically secure protocol for the above functionality is obtained by first implementing the information-theoretic OT combiner protocol from [CDFR17] to extract "pure" OT correlations from the above "leaky" OT correlations and then use the information-theoretic two-party protocols [Kil88, IPS08, IKO+11] in the OT correlations model to securely compute $x_\alpha \oplus r$. Unfortunately, this protocol does not run in two rounds. To squish the number of rounds, we apply the round collapsing compiler of [BL18, GS18] to this larger round protocol and use the protocol from the first step (the one that suffers from input dependent abort) to set up the leaky OT correlations. Since the above protocol is statistical, the squished protocol only makes black-box use of cryptographic operations. Additionally, to enable the party P_3 to output $y_{x_\alpha \oplus r}$, we use the following observation about the compiler given in [GS18]: even if a party is not participating in the protocol, the garbled circuit generated by the party can listen to the protocol transcript and thus, learn the output. This observation allows the garbled circuit generated by P_3 to listen to the protocol between P_1 and P_2 and obtain $x_\alpha \oplus r$. This garbled circuit can then output $y_{x_\alpha \oplus r}$. This allows us to obtain a three-round black-box protocol for the double selection functionality that does not suffer from input dependent abort.

From Double Selection to General Functions. To give a protocol for general functions, we use the techniques in [GIS18] to show that double selection is black-box complete for designing three-round secure protocols against malicious adversaries. Specifically, we apply the round-collapsing compiler to statistically secure protocols in the OT correlations model [Kil88, IPS08] and use the above protocol to implement the double selection functionality. This gives rise to a three-round MPC protocol that makes black-box use of a two-round, malicious-secure oblivious transfer with equivocal receiver security.

3 Preliminaries

We recall some standard cryptographic definitions in this section. Let λ denote the security parameter. A function $\mu(\cdot) : \mathbb{N} \rightarrow \mathbb{R}^+$ is said to be *negligible* if for any polynomial $\mathsf{poly}(\cdot)$, there exists $\lambda_0 \in \mathbb{N}$ such that for all $\lambda > \lambda_0$, we have $\mu(\lambda) < \frac{1}{\mathsf{poly}(\lambda)}$. We will use $\mathsf{negl}(\cdot)$ to denote an unspecified negligible function and $\mathsf{poly}(\cdot)$ to denote an unspecified polynomial function. We say that two distribution ensembles $X = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable (denoted by $X \stackrel{c}{\approx} Y$) if for every PPT distinguisher D there exists a negligible function $\mathsf{negl}(\cdot)$ such that, $|\Pr[D(X_\lambda) = 1] - \Pr[D(Y_\lambda) = 1]| \leq \mathsf{negl}(\lambda)$. We use $\stackrel{s}{\approx}$ to denote statistical indistinguishability and $\equiv$ to denote that the two distributions are identical. For a binary string x of length n, we use $x[k]$ to denote the k-th bit of the string x for some $k \in [n]$.

For a probabilistic algorithm A, we denote $A(x; r)$ to be the output of A on input x with the contents of the random tape being r. When r is omitted, $A(x)$ denotes a distribution. For a finite set S, we denote $x \leftarrow S$ as the process of sampling x uniformly from the set S. We will use PPT to denote Probabilistic Polynomial Time algorithm. By default, we allow our adversarial algorithms to take a polynomimal sized advice (i.e., non-uniform PPT algorithms).

3.1 Universal Composition Framework

We work in the the Universal Composition (UC) framework [Can01] to formalize and analyze the security of our protocols. Our protocols can also be analyzed in the stand-alone setting, using the composability framework of [Can00a], or in other UC-like frameworks, like that of [PW00]. We refer the reader to [Can00b] for the details on the UC framework and for completeness, we provide a brief overview in Appendix A.

3.2 Garbled Circuits

We recall the definition of garbling scheme for circuits [Yao86] (see Applebaum et al. [AIK04, App17], Lindell and Pinkas [LP09] and Bellare et al. [BHR12] for a detailed proof and further discussion). A garbling scheme for circuits is a tuple of PPT algorithms $(\mathsf{Garble}, \mathsf{Eval})$. Garble is the circuit garbling procedure and Eval is the corresponding evaluation procedure. More formally:

- $(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,b}\}_{w\in\mathsf{inp}(C),b\in\{0,1\}}) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{C}\right)$: Garble takes as input a security parameter 1^λ, a circuit C, and outputs a *garbled circuit* $\widetilde{\mathsf{C}}$ along with *labels* $\mathsf{lab}_{w,b}$ where $w \in \mathsf{inp}(C)$ ($\mathsf{inp}(C)$ is the set of input wires of C) and $b \in \{0, 1\}$. Each label $\mathsf{lab}_{w,b}$ is assumed to be in $\{0, 1\}^\lambda$.
- $y \leftarrow \mathsf{Eval}\left(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,x_w}\}_{w\in\mathsf{inp}(C)}\right)$: Given a garbled circuit $\widetilde{\mathsf{C}}$ and a sequence of input labels $\{\mathsf{lab}_{w,x_w}\}_{w\in\mathsf{inp}(C)}$ (referred to as the garbled input), Eval outputs a string y.

Correctness. For correctness, we require that for any circuit C and any input $x \in \{0, 1\}^{|\mathsf{inp}(C)|}$, we have that:

$$\Pr\left[\mathsf{C}(x) = \mathsf{Eval}\left(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,x_w}\}_{w\in\mathsf{inp}(C)}\right)\right] = 1$$

where $(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,b}\}_{w\in\mathsf{inp}(C),b\in\{0,1\}}) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{C}\right)$.

Security. For security, we require that there exists a PPT simulator $\mathsf{Sim}_{\mathsf{GC}}$ such that for any circuit C and any input $x \in \{0, 1\}^{|\mathsf{inp}(C)|}$, we have that:

$$\left(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,x_w}\}_{w\in\mathsf{inp}(C)}\right) \overset{c}{\approx} \mathsf{Sim}_{\mathsf{GC}}\left(1^{|C|}, 1^{|x|}, \mathsf{C}(x)\right)$$

where $(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,b}\}_{w\in\mathsf{inp}(C),b\in\{0,1\}}) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{C}\right)$.

Authenticity of Input labels. We require for any circuit C and input $x \in \{0, 1\}^{|\mathsf{inp}(C)|}$ and for any PPT adversary $\mathcal{A}$, the probability that the following game outputs 1 is negligible.

$$\begin{aligned}
\left(\widetilde{C}, \{\mathsf{lab}_w\}_{w\in\mathsf{inp}(C)}\right) &\leftarrow \mathsf{Sim}\left(1^{|C|}, 1^{|x|}, \mathsf{C}(x)\right) \\
\{\mathsf{lab}'_w\}_{w\in\mathsf{inp}(C)} &\leftarrow \mathcal{A}(\widetilde{C}, \{\mathsf{lab}_w\}_{w\in\mathsf{inp}(C)}) \\
y &= \mathsf{Eval}(\widetilde{C}, \{\mathsf{lab}'_w\}_{w\in\mathsf{inp}(C)}) \\
(\{\mathsf{lab}_w\}_{w\in\mathsf{inp}(C)} \neq \{\mathsf{lab}'_w\}_{w\in\mathsf{inp}(C)}) &\bigwedge (y \neq \bot)
\end{aligned}$$

Remark 3.1. *The authenticity of input labels property was needed in [GS18] to achieve security with unanimous abort. [GS18] noted that we can add this property to any garbled circuit construction by digitally signing the labels and including the signatures along with the labels and including the verification key along with the garbled circuit $\widetilde{C}$. The evaluation procedure first checks the signatures before proceeding with the actual evaluation of garbled circuit.*

3.3 Oblivious Transfer

In this paper, we consider a 1-out-of-2 *oblivious transfer* protocol (OT), similar to [CCM98, NP01, AIR01, DHRS04, PVW08, HK12] where one party, the *sender*, has input composed of two strings (s_0, s_1) and the input of the second party, the *receiver*, is a bit β. The receiver should learn s_β and nothing regarding $s_{1-\beta}$, while the sender should gain no information about β.

Semi-Honest Secure Two-Round Oblivious Transfer. A two-round semi-honest OT protocol $\langle S, R\rangle$ is defined by three probabilistic algorithms $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ as follows. The receiver runs the algorithm OT_1 with the security parameter 1^λ, and a bit $\beta \in \{0,1\}$ as input and the random tape set to ω and obtains otr. The receiver then sends otr to the sender, who obtains ots by evaluating $\mathsf{OT}_2(\mathsf{otr}, (s_0, s_1))$ (with a uniform random tape), where $s_0, s_1 \in \{0,1\}^\lambda$ are the sender's input messages. The sender then sends ots to the receiver who obtains s_β by evaluating $\mathsf{OT}_3(\mathsf{ots}, (\beta, \omega))$.

- **Correctness.** For every choice bit $\beta \in \{0,1\}$ and the random tape ω of the receiver, and any input messages s_0 and s_1 of the sender we require that, if $\mathsf{otr} := \mathsf{OT}_1(1^\lambda, \beta; \omega)$, $\mathsf{ots} \leftarrow \mathsf{OT}_2(\mathsf{otr}, (s_0, s_1))$, then $\mathsf{OT}_3(\mathsf{ots}, (\beta, \omega)) = s_\beta$ with probability 1.

- **Receiver's security.** We require that,

$$\left\{\mathsf{otr} : \omega \leftarrow \{0,1\}^*, \mathsf{otr} := \mathsf{OT}_1(1^\lambda, 0; \omega)\right\} \overset{c}{\approx} \left\{\mathsf{otr} : \omega \leftarrow \{0,1\}^*, \mathsf{otr} := \mathsf{OT}_1(1^\lambda, 1; \omega)\right\}.$$

- **Sender's security.** We require that for any choice of $\beta \in \{0,1\}$ and any strings $K_0, K_1, L_0, L_1 \in \{0,1\}^\lambda$ with $L_0 = L_1 = K_\beta$, we have that,

$$\left\{\beta, \omega \leftarrow \{0,1\}^*, \mathsf{OT}_2(1^\lambda, \mathsf{otr}, K_0, K_1)\right\} \overset{c}{\approx} \left\{\beta, \omega \leftarrow \{0,1\}^*, \mathsf{OT}_2(1^\lambda, \mathsf{otr}, L_0, L_1)\right\}$$

  where $\mathsf{otr} := \mathsf{OT}_1(1^\lambda, \beta; \omega)$.

Remark 3.2. *We note that we can relax the correctness requirement to have a negligible probability of error. For the sake of simplicity of exposition, we stick to protocols having perfect correctness.*

Maliciously Secure Two-Round Oblivious Transfer with Equivocal Receiver Security. We consider the stronger notion of oblivious transfer with security against malicious adversaries in the common random/reference string model. In addition to the standard security against malicious receivers, we need this protocol to satisfy a special property called equivocal receiver security introduced in [GS18]. Informally, this property says that the first round message of the receiver can be equivocated to both choice bits 0 and 1. In terms of syntax, we supplement the syntax of semi-honest oblivious transfer with an algorithm K_{OT} that takes the security parameter 1^λ as input and outputs the common random/reference string crs. Also, the three algorithms $\mathsf{OT}_1, \mathsf{OT}_2$ and OT_3 additionally take crs as input. Furthermore, instead of using the entire random tape of OT_1 algorithm as input to OT_3, we let the OT_1 algorithm to output some secret information which is then used by OT_3.

- **Correctness.** For every choice bit $\beta \in \{0,1\}$ and any input messages s_0 and s_1 of the sender, we require that, if $\mathsf{crs} \leftarrow K_{\mathsf{OT}}(1^\lambda)$, $(\mathsf{otr}, \mu) \leftarrow \mathsf{OT}_1(\mathsf{crs}, \beta)$, $\mathsf{ots} \leftarrow \mathsf{OT}_2(\mathsf{crs}, \mathsf{otr}, (s_0, s_1))$, then $\mathsf{OT}_3(\mathsf{crs}, \mathsf{ots}, (\beta, \mu)) = s_\beta$ with probability 1.

- **Equivocal Receiver's security.** We require the existence of a PPT simulator $\mathsf{Sim}_R = (\mathsf{Sim}_R^1, \mathsf{Sim}_R^2)$ such that for any sequence of $(\beta_1, \ldots, \beta_n)$ where each $\beta_i \in \{0,1\}$ and $n = \mathsf{poly}(\lambda)$, we have:

$$\left\{(\mathsf{crs}, \{(\mathsf{otr}^i, \mu^i_{\beta_i})\}_{i\in[n]}) : (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_R^1(1^\lambda), \{(\mathsf{otr}^i, \mu^i_0, \mu^i_1) \leftarrow \mathsf{Sim}_R^2(\mathsf{crs}, \mathsf{td})\}_{i\in[n]}\right\} \overset{c}{\approx} \left\{(\mathsf{crs}, \{\mathsf{OT}_1(\mathsf{crs}, \beta_i)\}_{i\in[n]}) : \mathsf{crs} \leftarrow K_{\mathsf{OT}}(1^\lambda)\right\}.$$

- **Checking Validity of Receiver's Key.** There is a deterministic polynomial time algorithm $\mathsf{CheckValid}$ that takes as input $\mathsf{crs}, \mathsf{otr}, \beta, \mu$ and outputs 1 if and only if there exists some $\omega \in \{0,1\}^*$ such that $(\mathsf{otr}, \mu) := \mathsf{OT}_1(\mathsf{crs}, \beta; \omega)$.

- **Sender's security.** We require the existence of PPT algorithm $\mathsf{Sim}_S = (\mathsf{Sim}_S^1, \mathsf{Sim}_S^2)$ such that for any choice of $K_0^i, K_1^i \in \{0,1\}^\lambda$ for $i \in [n]$ where $n = \mathsf{poly}(\lambda)$, PPT adversary $\mathcal{A}$ and any PPT distinguisher D, we have:

$$\left|\Pr[\mathrm{IND}^{\mathrm{REAL}}_{S,\mathcal{A},D}(1^\lambda, \{K_0^i, K_1^i\}_{i\in[n]}) = 1] - \Pr[\mathrm{IND}^{\mathrm{IDEAL}}_{S,\mathcal{A},D}(1^\lambda, \{K_0^i, K_1^i\}_{i\in[n]}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Experiment $\mathrm{IND}^{\mathrm{REAL}}_{S,\mathcal{A},D}(1^\lambda, \{K_0^i, K_1^i\}_{i\in[n]})$:	**Experiment** $\mathrm{IND}^{\mathrm{IDEAL}}_{S,\mathcal{A},D}(1^\lambda, \{K_0^i, K_1^i\}_{i\in[n]})$:
$\mathsf{crs} \leftarrow K_{\mathsf{OT}}(1^\lambda)$	$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_S^1(1^\lambda)$
$\{\mathsf{otr}^i\}_{i\in[n]} \leftarrow \mathcal{A}(\mathsf{crs})$	$\{\mathsf{otr}^i\}_{i\in[n]} \leftarrow \mathcal{A}(\mathsf{crs})$
	$\beta_i := \mathsf{Sim}_S^2(\mathsf{crs}, \mathsf{td}, \mathsf{otr}^i)\ \forall i \in [n]$
	$L_0^i := K^i_{\beta_i}$ and $L_1^i := K^i_{\beta_i}$
$\mathsf{ots}^i \leftarrow \mathsf{OT}_2(\mathsf{crs}, \mathsf{otr}^i, (K_0^i, K_1^i)),\ \forall i \in [n]$	$\mathsf{ots}^i \leftarrow \mathsf{OT}_2(\mathsf{crs}, \mathsf{otr}, (L_0^i, L_1^i)),\ \forall i \in [n]$
Output $D(\mathsf{crs}, \{\mathsf{ots}^i\}_{i\in[n]})$	Output $D(\mathsf{crs}, \{\mathsf{ots}^i\}_{i\in[n]})$

Remark 3.3. *We note that a two-round malicious secure oblivious transfer with equivocal receiver security implies a standard two-round malicious oblivious transfer that implements the ideal OT functionality.*

Remark 3.4. *A two-round malicious-secure oblivious transfer with equivocal receiver security can be instantiated from black-box use of any of the following primitives:*

- *A two-round malicious-secure oblivious transfer that is secure against an adversary that can adaptively corrupt the receiver [CFGN96, CLOS02].*

- *A dual-mode public key encryption scheme [PVW08].*

3.4 Non-Interactive Secure Computation

We now recall the notion of non-interactive secure computation introduced in the works of Ishai et al. [IPS08, IKO+11]. A non-interactive secure computation Π_{NISC} for a function f (that is possibly randomized) is a two-party protocol between a receiver and a sender and is given by a tuple of algorithms $(K_{\mathsf{NISC}}, \mathsf{NISC}_1, \mathsf{NISC}_2, \mathsf{NISC}_3)$. The algorithm K_{NISC} is a common random/reference string generating algorithm that takes the security parameter 1^λ (encoded in unary) and samples crs from some pre-defined distribution. The receiver runs NISC_1 algorithm on crs and its input x

(with a random tape ω) to get msg_1 which it sends to the sender. The sender runs NISC_2 on crs, the message msg_1 sent by the receiver and its own input y to get msg_2 which it sends to the receiver. The receiver runs NISC_3 on crs, the sender's message msg_2 and (x, ω) to get the output $f(x, y)$. The works of Ishai et al. [IPS08, IKO+11] showed the following theorem.

Theorem 3.5 ([IPS08, IKO+11])**.** *Let f be any (possibly randomized) two-party functionality. There exists a non-interactive secure computation protocol Π_{NISC} that UC-realizes $\mathcal{F}_f$ making black-box access to a two-round, malicious-secure oblivious transfer.*

Rabin-OT functionality. In this work, we will be interested in a non-interactive secure computation protocol that securely realizes the Rabin OT functionality $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$. The (m, p)-Rabin OT is a randomized functionality takes m strings $s_1, \ldots, s_m$ from the sender and for each $i \in [m]$, it independently replaces s_i with s'_i where $s'_i = s_i$ with probability p and $s'_i = \bot$ with probability $1 - p$. It then outputs $(s'_1, \ldots, s'_m)$ to the receiver. This functionality is formally given in Fig. 3.

3.5 Bivariate Polynomials

In this subsection, we recall some simple facts about (symmetric) bivariate polynomials. Let $\mathbb{F}$ be a finite field such that $|\mathbb{F}| > n$ for some $n \in \mathbb{N}$. Let $\alpha_1, \ldots, \alpha_n$ be distinct non-zero elements from $\mathbb{F}$.

Definition 3.6. *A bivariate polynomial over $\mathbb{F}$ with degree t is a polynomial over two variables such that the degree of both these variables is t. Such a polynomial can be expressed as: $f(x, y) = \sum_{i=0}^{i=t} \sum_{j=0}^{j=t} c_{i,j} x^i y^j$ where $c_{i,j} \in \mathbb{F}$. In a symmetric bivariate polynomial $c_{i,j} = c_{j,i}$ for all $i, j \in [0, t]$*

We now recall the following two standard facts about symmetric bivariate polynomials.

Fact 3.7. *Let $K \subseteq [n]$ be a set of indices such that $|K| \geq t + 1$, let $\{f_k(x)\}_{k \in K}$ be a set of degree-t polynomials over $\mathbb{F}$. If for every $i, j \in K$, it holds that $f_i(\alpha_j) = f_j(\alpha_i)$, then there exists a unique symmetric bivariate polynomial S of degree-t over $\mathbb{F}$ such that $f_k(x) = S(x, \alpha_k)$ for every $k \in K$.*

Fact 3.8. *Let $I \subseteq [n]$ be a set of indices such that $|I| \leq t$. For two elements $s_1, s_2 \in \mathbb{F}$, let S_1 and S_2 be random symmetric degree-t bivariate polynomials such that $S_1(0, 0) = s_1$ and $S_2(0, 0) = s_2$. Then,*

$$\{(i, S_1(x, \alpha_i))\}_{i \in I} \equiv \{(i, S_2(x, \alpha_i))\}_{i \in I}$$

4 3-Round Semi-Honest MPC

In this section, we give a three-round, semi-honest secure protocol for computing arbitrary multiparty functionalities making black-box use of a two-round, semi-honest secure oblivious transfer in the plain model. We do this in two steps. In the first step, we give a three round protocol for securely computing the $\mathcal{F}_{\mathsf{3MULTPlus}}$ functionality (described below) against semi-honest adversaries. In the second step, we use the results from [BGI+18, GIS18, ABG+20] to extend this to securely compute general functions.

4.1 Step-1: Protocol for $\mathcal{F}_{\mathsf{3MULTPlus}}$

Let us first recall the $\mathcal{F}_{\mathsf{3MULTPlus}}$ functionality. It is a n-party functionality that takes input from 3 parties and delivers output to every party. Specifically, let us denote the parties that provide inputs to this functionality by P_1, P_2, and P_3. The input of P_i for $i \in \{1, 2, 3\}$ is given by $(x_i, y_i) \in \{0, 1\} \times \{0, 1\}$. The output of the functionality is given by $x_1 \cdot x_2 \cdot x_3 + y_1 + y_2 + y_3$ (where $+$ and $\cdot$ are over $\mathbb{F}_2$). The main theorem that we show in this subsection is:

Theorem 4.1. *There is an efficient three-round protocol that makes black-box use of a two-round, semi-honest oblivious transfer and securely computes the $\mathcal{F}_{\mathsf{3MULTPlus}}$ functionality against semi-honest adversaries corrupting an arbitrary subset of the parties. The protocol is in the plain model.*

Building $\Pi_{\mathsf{3MULTPlus}}$. In Figure 1, we give the formal description of the protocol and provide an informal overview below.

At a high-level, the degree-3 computation in the $\mathcal{F}_{\mathsf{3MULTPlus}}$ functionality is achieved by cascading OT messages i.e., making a sender OT message to include two other sender OT message as its inputs. Since OT enables degree-2 computation, cascading OT brings the desired result of a degree-3 computation. The innovation lies in being able to do this in 2 rounds for OTs that are run in parallel. The last round is spent on a single broadcast of a value by each party and subsequent local accumulation of these broadcasted values to obtain the final result. We elaborate on this idea below. P_1, acting as a receiver, publishes an OT receiver message otr for its input x_1. In parallel, P_2, acting as a receiver, publishes two OT receiver messages, $\mathsf{otr}_0, \mathsf{otr}_1$ for two (additive) shares $x_{2,0}, x_{2,1}$ of its input x_2. In the second round. P_3 splits its input x_3 into two additive shares, $x_{3,0}, x_{3,1}$, and prepares two OT sender messages with respect to the receiver messages $\mathsf{otr}_0, \mathsf{otr}_1$ using $(x_{3,0}, x_{3,1})$ as the input in both the messages. Let these be denoted by $\mathsf{ots}_0, \mathsf{ots}_1$. The crux of our construction is then to use $\mathsf{ots}_0, \mathsf{ots}_1$ as a sender input in response to P_1's receiver message otr. With this sender message, P_1 can retrieve ots_{x_1}, but in order to decode ots_{x_1}, it needs the receiver's input and randomness used for ots_{x_1}, which are held by P_2. Responding to P_1's receiver message otr, P_2 computes a sender OT message with input $((x_{2,0}, \omega_{2,0}), (x_{2,1}, \omega_{2,1}))$. Using this message, P_1 can retrieve x_{2,x_1} and the corresponding randomness while $x_{2,1-x_1}$ and the matching randomness are hidden. Deducing from the OT correctness, we can now conclude that P_1 in the end of the computation receives $x_{3,x_{2,x_1}}$ which can be written as $x_{2,x_1}(x_{3,0}+x_{3,1})+x_{3,0} = (x_1 \cdot x_2 + x_{2,0}) \cdot x_3 + x_{3,0}$, since $x_{2,x_1} = x_1(x_{2,0} + x_{2,1}) + x_{2,0}$. To cancel out the extra multiplicative term $x_{2,0} \cdot x_3$ in the expression, another OT instance is needed between P_2, P_3, where P_3 enacts a receiver with input x_3 and P_2 enacts a sender with input $x_{2,0,0}, x_{2,0,1}$, two additive shares of $x_{2,0}$. Once all the OTs conclude in the first two rounds, each of P_1, P_2 and P_3 accumulates their appropriate local data (which includes their other input y_i) and broadcasts. These broadcasts enable every party to compute the final result via plain addition. Lastly, each of these three parties distributes shares of 0 amongst P_1, P_2, P_3 to be added to their local sum before broadcast. This step is required for simulation for the case of more than one honest parties in the quorum P_1, P_2, P_3.

Protocol $\Pi_{\mathsf{3MULTPlus}}$

Inputs: P_i for $i \in [3]$ inputs (x_i, y_i).

Output: For each $i \in [n]$, P_i outputs $x_1x_2x_3 + y_1 + y_2 + y_3$.

Primitive: A two-round semi-honest secure oblivious transfer protocol defined by $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$.

Round-1: In the first round,

- P_1 chooses a random string $\omega \leftarrow \{0,1\}^*$ and computes $\mathsf{otr} := \mathsf{OT}_1(1^\lambda, x_1; \omega)$.
- P_2 chooses two random strings $\omega_0, \omega_1 \leftarrow \{0,1\}^*$. It chooses random bits $x_{2,0}, x_{2,1} \leftarrow \{0,1\}$ subject to $x_2 = x_{2,1} + x_{2,0}$. It computes $\mathsf{otr}_0 := \mathsf{OT}_1(1^\lambda, x_{2,0}; \omega_0)$ and $\mathsf{otr}_1 := \mathsf{OT}_1(1^\lambda, x_{2,1}; \omega_1)$.
- P_3 chooses a random string $\omega' \leftarrow \{0,1\}^*$ and computes $\mathsf{otr}_3 := \mathsf{OT}_1(1^\lambda, x_3; \omega')$.
- P_1 broadcasts otr, P_2 broadcasts $(\mathsf{otr}_0, \mathsf{otr}_1)$ and P_3 broadcasts otr_3.
- For every $i \in [3]$, P_i chooses a random additive secret sharing of 0 given by $(\delta_1^i, \delta_2^i, \delta_3^i)$ and sends the share δ_j^i to party P_j for $j \in [3] \setminus \{i\}$ via private channels. We note that we can simulate a single round of private channel messages in two-rounds over public channels by making use of a two-round oblivious transfer.

Round-2: In the second round,

- P_2 computes $\mathsf{ots} \leftarrow \mathsf{OT}_2(\mathsf{otr}, (x_{2,0}, \omega_0), (x_{2,1}, \omega_1))$. It then chooses random bits $x_{2,0,0}, x_{2,0,1} \leftarrow \{0,1\}$ subject to $x_{2,0} = x_{2,0,0} + x_{2,0,1}$. It computes $\mathsf{ots}_3 \leftarrow \mathsf{OT}_2(\mathsf{otr}_3, x_{2,0,0}, x_{2,0,1})$.
- P_3 chooses random bits $x_{3,0}, x_{3,1} \leftarrow \{0,1\}$ subject to $x_3 = x_{3,0} + x_{3,1}$. For each $b \in \{0,1\}$, it first computes $\mathsf{ots}_b \leftarrow \mathsf{OT}_2(\mathsf{otr}_b, x_{3,0}, x_{3,1})$. It then computes $\overline{\mathsf{ots}} \leftarrow \mathsf{OT}_2(\mathsf{otr}, \mathsf{ots}_0, \mathsf{ots}_1)$.
- P_2 broadcasts $(\mathsf{ots}, \mathsf{ots}_3)$ and P_3 broadcasts $\overline{\mathsf{ots}}$.

Round-3: In the last round,

- For each $i \in [3]$, P_i computes $\delta_i = \delta_i^1 + \delta_i^2 + \delta_i^3$.
- P_2 sets $z_2 := x_{2,0,0} + y_2 + \delta_2$.
- P_3 computes $x_{2,0,x_3} := \mathsf{OT}_3(\mathsf{ots}_3, (x_3, \omega'))$ and sets $z_3 = x_{2,0,x_3} + x_{3,0} + y_3 + \delta_3$.
- P_1 computes $(x_{2,x_1}, \omega_{x_1}) := \mathsf{OT}_3(\mathsf{ots}, (x_1, \omega))$ and $\mathsf{ots}_{x_1} := \mathsf{OT}_3(\overline{\mathsf{ots}}, (x_1, \omega))$. It then computes $x_{3,x_{2,x_1}} := \mathsf{OT}_3(\mathsf{ots}_{x_1}, (x_{2,x_1}, \omega_{x_1}))$. It then sets $z_1 := x_{3,x_{2,x_1}} + y_1 + \delta_1$.
- P_1 broadcasts z_1, P_2 broadcasts z_2 and P_3 broadcasts z_3.

Output: Every party outputs $z_1 + z_2 + z_3$.

Figure 1: Protocol $\Pi_{\mathsf{3MULTPlus}}$

In Lemma 4.2, we show the correctness and in Lemma 4.3, we show the security of the protocol. We start with the correctness proof.

Lemma 4.2 (Correctness)**.** *Protocol $\Pi_{\mathsf{3MULTPlus}}$ (Figure 1) correctly computes the $\mathcal{F}_{\mathsf{3MULTPlus}}$ functionality.*

Proof. We first observe that $x_{2,0,x_3}$ computed by P_3 in Round-3 is equal to $x_3(x_{2,0,0}+x_{2,0,1})+x_{2,0,0} = x_3 \cdot x_{2,0} + x_{2,0,0}$. Therefore, $z_3 = x_3 \cdot x_{2,0} + x_{2,0,0} + x_{3,0} + y_3 + \delta_3$. We then observe that x_{2,x_1} and ots_{x_1} computed by P_1 are equal to $x_1 \cdot x_2 + x_{2,0}$ and $\mathsf{OT}_2(\mathsf{OT}_1(1^\lambda, x_{2,x_1}; \omega_{x_1}), x_{3,0}, x_{3,1})$ respectively. Therefore, $x_{3,x_{2,x_1}}$ computed by P_1 is equal to $x_{2,x_1}(x_{3,0} + x_{3,1}) + x_{3,0} = (x_1 \cdot x_2 + x_{2,0}) \cdot x_3 + x_{3,0}$. This implies that $z_1 = (x_1 \cdot x_2 + x_{2,0}) \cdot x_3 + x_{3,0} + y_1 + \delta_1$. Finally, we observe that $(\delta_1, \delta_2, \delta_3)$ form an additive secret sharing of 0. Hence,

$$\begin{aligned} z_1 + z_2 + z_3 &= ((x_1 \cdot x_2 + x_{2,0}) \cdot x_3 + x_{3,0} + y_1 + \delta_1) \\ &+ (x_{2,0,0} + y_2 + \delta_2) + (x_3 \cdot x_{2,0} + x_{2,0,0} + x_{3,0} + y_3 + \delta_3) \\ &= x_1 \cdot x_2 \cdot x_3 + y_1 + y_2 + y_3 \end{aligned}$$

This completes the proof of correctness. □

We now show the semi-honest security of the protocol.

Lemma 4.3 (Security). *Protocol $\Pi_{\mathsf{3MULTPlus}}$ (Figure 1) securely computes $\mathcal{F}_{\mathsf{3MULTPlus}}$ functionality against a semi-honest adversary corrupting an arbitrary subset of parties.*

Proof. Let $\mathcal{A}$ be a semi-honest adversary against the protocol. Let C be the set of parties corrupted by $\mathcal{A}$ and let H be the set of honest parties. If $\{P_1, P_2, P_3\} \subseteq C$, then the description of the simulator is trivial and hence, we assume that $\{P_1, P_2, P_3\} \cap H \neq \varnothing$. We now describe a simulator Sim that simulates the view of the adversary $\mathcal{A}$.

Interaction with environment $\mathcal{Z}$. For every input value corresponding to the set of corrupted parties C that Sim receives from the environment $\mathcal{Z}$, Sim writes this value to $\mathcal{A}$'s input tape. Similarly, the contents of $\mathcal{A}$'s output tape is written to Sim's output tape. We now describe how Sim simulates the interaction of honest parties with $\mathcal{A}$.

Simulating the interaction with $\mathcal{A}$: For every concurrent interaction with the session identifier sid that $\mathcal{A}$ may start, the simulator does the following:

- **Initialization.** Sim sets the random tape of $\mathcal{A}$ with a uniformly chosen string and starts the interaction with $\mathcal{A}$.
- **Round-1 messages from Sim to $\mathcal{A}$:**
  - If $P_1 \in H$, then Sim chooses a random string $\omega \leftarrow \{0,1\}^*$ and sets $\underline{\mathsf{otr} := \mathsf{OT}_1(1^\lambda, 0; \omega)}$.
  - If $P_2 \in H$, then Sim chooses a random bit $x_{2,0} \leftarrow \{0,1\}$ and sets $\underline{x_{2,1} = x_{2,0}}$. It then chooses two random strings $\omega_0, \omega_1 \leftarrow \{0,1\}^*$ and computes $\mathsf{otr}_0 := \mathsf{OT}_1(1^\lambda, x_{2,0}; \omega_0)$ and $\mathsf{otr}_1 := \mathsf{OT}_1(1^\lambda, x_{2,1}; \omega_1)$.
  - If $P_3 \in H$, Sim chooses a random string $\omega' \leftarrow \{0,1\}^*$ and sets $\underline{\mathsf{otr}_3 := \mathsf{OT}_1(1^\lambda, 0; \omega')}$.
  - For each $P_i \in C \cap \{P_1, P_2, P_3\}$, Sim sends a random bit on behalf of every $P_j \in H \cap \{P_1, P_2, P_3\}$ to P_i through private channels.
  - Sim sends the rest of the first round messages on behalf of the honest parties to $\mathcal{A}$.
- **Round-2 messages from Sim to $\mathcal{A}$:**
  - If $P_2 \in H$, then Sim does the following:
    - Sim sets $\mathsf{ots} \leftarrow \mathsf{OT}_2(\mathsf{otr}, (x_{2,0}, \omega_0), (x_{2,1}, \omega_1))$.
    - It then chooses a random bit $x_{2,0,0} \leftarrow \{0,1\}$ and sets $\underline{x_{2,0,1} = x_{2,0,0}}$. It then computes $\mathsf{ots}_3 \leftarrow \mathsf{OT}_2(\mathsf{otr}_3, x_{2,0,0}, x_{2,0,1})$.

- If $P_3 \in H$, then Sim does the following:
  - Sim chooses a random bit $x_{3,0}$ and sets $\underline{x_{3,1} := x_{3,0}}$.
  - For each $b \in \{0,1\}$, it first computes $\mathsf{ots}_b \leftarrow \mathsf{OT}_2(\mathsf{otr}_b, x_{3,0}, x_{3,1})$. It then computes $\overline{\mathsf{ots}} \leftarrow \mathsf{OT}_2(\mathsf{otr}, \mathsf{ots}_0, \mathsf{ots}_1)$.
- Sim sends the second round messages on behalf of the honest parties to $\mathcal{A}$.

- **Round-3 messages from Sim to $\mathcal{A}$.**
  - For every $P_i \in C \cap \{P_1, P_2, P_3\}$, Sim computes z_i by using the messages sent in the first two rounds, the input (x_i, y_i) (received from the environment) and the random tape of party P_i (that it set).
  - Let z be the output of the $\mathcal{F}_{\mathsf{3MULTPlus}}$ functionality (obtained by Sim by querying the ideal functionality).
  - For every $P_i \in H \cap \{P_1, P_2, P_3\}$, Sim chooses z_i uniformly at random from $\{0,1\}$ subject to $\bigoplus_{i \in H \cap \{P_1,P_2,P_3\}} z_i = z \oplus \left(\bigoplus_{i \in C \cap \{P_1,P_2,P_3\}} z_i \right)$.
  - Sim sends $\{z_i\}_{i \in H \cap \{P_1,P_2,P_2\}}$ to $\mathcal{A}$.

Proof of Indistinguishability. We now show that the simulated interaction is indistinguishable to the real world interaction via a hybrid argument.

- $\underline{\mathsf{Hybrid}_0}$: This corresponds to the view of the adversary and the outputs of the honest parties in the real world execution of the protocol.

- $\underline{\mathsf{Hybrid}_1}$: Skip this hybrid change if $P_1 \notin H$. In this hybrid, we set otr sent by P_1 in the first round to be equal to $\mathsf{OT}_1(1^\lambda, 0; \omega)$ instead of $\mathsf{OT}_1(1^\lambda, x_1; \omega)$. In round-3, instead of using the OT_3 computation, we use the knowledge of inputs and random tape of P_2 and P_3 to recover $(x_{2,x_1}, \omega_{x_1})$ and ots_{x_1} respectively.

- $\underline{\mathsf{Hybrid}_2}$: Skip this hybrid if $P_3 \notin H$. In this hybrid, we set otr_3 sent by P_3 in the first round to be equal to $\mathsf{OT}_1(1^\lambda, 0; \omega')$ instead of $\mathsf{OT}_1(1^\lambda, x_3; \omega')$. In round-3, instead of using the OT_3 computation, we use the knowledge of inputs and random tape of P_2 to recover $x_{2,0,x_3}$.

- $\underline{\mathsf{Hybrid}_3}$: Skip this hybrid if $P_2 \notin H$. In this hybrid, we set $\mathsf{ots} \leftarrow \mathsf{OT}_2(\mathsf{otr}, (x_{2,x_1}, \omega_{x_1}), (x_{2,x_1}, \omega_{x_1}))$ instead of $\mathsf{OT}_2(\mathsf{otr}, (x_{2,0}, \omega_0), (x_{2,1}, \omega_1))$. Similarly, we set ots_3 to be equal to $\mathsf{OT}_2(\mathsf{otr}_3, x_{2,0,x_3}, x_{2,0,x_3})$ instead of $\mathsf{OT}_2(\mathsf{otr}_3, x_{2,0,0}, x_{2,0,1})$.

- $\underline{\mathsf{Hybrid}_4}$: Skip this hybrid if $P_2 \notin H$. In this hybrid, we set otr_{1-x_1} to be $\mathsf{OT}_1(1^\lambda, x_{2,x_1}; \omega_{1-x_1})$ instead of $\mathsf{OT}_1(1^\lambda, x_{2,1-x_1}; \omega_{1-x_1})$. Here, we are making use of the observation that ω_{1-x_1} is not needed to simulate the other messages.

- $\underline{\mathsf{Hybrid}_5}$: Skip this hybrid if $P_2 \notin H$. In this hybrid, we set $x_{2,1-x_1} = x_{2,x_1}$ and $x_{2,0,1-x_3} = x_{2,0,x_3}$. Notice that this is only a syntactic change and hence, Hybrid_4 is identically distributed to Hybrid_5.

- $\underline{\mathsf{Hybrid}_6}$: Skip this hybrid if $P_3 \notin H$. In this hybrid, for each $b \in \{0,1\}$, we set $\mathsf{ots}_b \leftarrow \mathsf{OT}_2(\mathsf{otr}_b, x_{3,x_{2,b}}, x_{3,x_{2,b}})$ instead of $\mathsf{OT}_2(\mathsf{otr}_b, x_{3,0}, x_{3,1})$. We then set $\overline{\mathsf{ots}} \leftarrow \mathsf{OT}_2(\mathsf{otr}, \mathsf{ots}_{x_1}, \mathsf{ots}_{x_1})$. Thus, to generate $\overline{\mathsf{ots}}$ we only make use of $x_{3,x_{2,x_1}}$.

- $\underline{\mathsf{Hybrid}_7}$: Skip this hybrid if $P_3 \notin H$. In this hybrid, we set $x_{3,1-x_{2,x_1}} = x_{3,x_{2,x_1}}$. Note that this change is only syntactic and hence, Hybrid_7 is identically distributed to Hybrid_6.

- $\underline{\mathsf{Hybrid}_8}$: Let i^* be the smallest integer such that $P_{i^*} \in H \cap \{P_1, P_2, P_3\}$. In this hybrid, we set $z_{i^*} = z - \sum_{j \in [3] \setminus \{i^*\}} z_j$ instead of computing it as in the previous hybrid.

- $\underline{\mathsf{Hybrid}_9}$: For every $i \in H \cap \{P_1, P_2, P_3\}$ and $i \neq i^*$, we choose z_i uniformly at random.

- $\underline{\mathsf{Hybrid}_{10}}$: Skip this hybrid if $P_2 \notin H$. In this hybrid, we reset $\mathsf{ots} \leftarrow \mathsf{OT}_2(\mathsf{otm}_1, (x_{2,0}, \omega_0), (x_{2,1}, \omega_1))$ instead of $\mathsf{OT}_2(\mathsf{otr}, (x_{2,x_1}, \omega_{x_1}), (x_{2,x_1}, \omega_{x_1}))$. Note that since we have set $x_{2,0} = x_{2,1}$, the only difference between the two pairs of sender inputs is in the second component (i.e., the ω part).

- $\underline{\mathsf{Hybrid}_{11}}$: Skip this hybrid if $P_3 \notin H$. In this hybrid, we reset for each $b \in \{0,1\}$, $\mathsf{ots}_b = \mathsf{OT}_2(\mathsf{otr}_b, x_{3,0}, x_{3,1})$ and $\overline{\mathsf{ots}} \leftarrow \mathsf{OT}_2(\mathsf{otr}, \mathsf{ots}_0, \mathsf{ots}_1)$. Note that we have set $x_{3,0} = x_{3,1} = x_{3,x_{2,x_1}}$.

- $\underline{\mathsf{Hybrid}_{12}}$: This hybrid is identically distributed to the simulated interaction.

We now argue that for every $i \in [12]$, Hybrid_i is computationally indistinguishable from Hybrid_{i-1} using the security of the two-round oblivious transfer.

Claim 4.4. *Assuming the receiver security of the two-round semi-honest oblivious transfer, we have* $\mathsf{Hybrid}_0 \overset{c}{\approx} \mathsf{Hybrid}_1$.

Proof. Assume for the sake of contradiction that the adversary $\mathcal{A}$ can distinguish between the outputs of Hybrid_0 and Hybrid_1 with non-negligible advantage. We will use this adversary to construct an adversary $\mathcal{B}$ that breaks the receiver security of oblivious transfer.

$\mathcal{B}$ interacts with the challenger against the receiver OT security by giving 0 and x_1 as the challenge bits. It receives otr from the external challenger. It uses the received otr as the message from honest P_1 in the first round. In the last round, $\mathcal{B}$ uses the knowledge of the inputs and the random tape of the other parties to recover $(x_{2,x_1}, \omega_{x_1})$ and ots_{x_1}. It then computes $x_{3,x_{2,x_1}} := \mathsf{OT}_3(\mathsf{ots}_{x_1}, (x_{2,x_1}, \omega_{x_1}))$ and sets $z_1 := x_{3,x_{2,x_1}} + y_1 + \delta_1$. It broadcasts z_1 in the final round.

Notice that if otr contains the choice bit x_1, then the view of $\mathcal{A}$ along with the outputs of the honest parties is identical to Hybrid_0. Else, it is identically distributed to Hybrid_1. Thus, if $\mathcal{A}$ can distinguish between Hybrid_0 and Hybrid_1 with non-negligible advantage, then $\mathcal{B}$ can break the receiver security of the two-round oblivious transfer with non-negligible advantage which is a contradiction. □

Claim 4.5. *Assuming the receiver security of the two-round oblivious transfer, we have* $\mathsf{Hybrid}_1 \overset{c}{\approx} \mathsf{Hybrid}_2$.

Proof. The proof of this claim is similar to Claim 4.4. □

Claim 4.6. *Assuming the sender security of two-round oblivious transfer, we have* $\mathsf{Hybrid}_2 \overset{c}{\approx} \mathsf{Hybrid}_3$.

Proof. Notice that if P_1 is not corrupted then it directly follows from the security of oblivious transfer in the no corruption setting that $\mathsf{OT}_2(\mathsf{otr}, (x_{2,x_1}, \omega_{x_1}), (x_{2,x_1}, \omega_{x_1}))$ is computationally indistinguishable to $\mathsf{OT}_2(\mathsf{otr}, (x_{2,0}, \omega_0), (x_{2,1}, \omega_1))$. A similar argument can be made for the case where P_3 is not corrupted. Hence, in the rest of the proof, we assume that both P_1 and P_3 are corrupted.

Assume for the sake of contradiction that the adversary $\mathcal{A}$ can distinguish between the outputs of Hybrid_2 and Hybrid_3 with non-negligible advantage. We will use $\mathcal{A}$ to construct another adversary $\mathcal{B}$ that can break the sender security of two-round oblivious transfer.

$\mathcal{B}$ sends x_1 as the receiver's choice bit and $(x_{2,0}, \omega_0)$, $(x_{2,1}, \omega_1)$ as the sender's input messages for the first oblivious transfer and sends x_3 as the receiver's choice bit and $(x_{2,0,0}, x_{2,0,1})$ as the sender's input messages for the second oblivious transfer. It receives $(x_1, \omega, \mathsf{ots})$ as the first challenge and $(x_3, \omega', \mathsf{ots}_3)$ as the second challenge. It sets the appropriate parts of the random tape of P_1 and P_3 to be ω and ω' respectively and begins the interaction with $\mathcal{A}$. In the second round, it uses the received challenge string $(\mathsf{ots}, \mathsf{ots}_3)$ as the messages sent by honest P_2. The rest of the steps are identical to the previous hybrid.

Notice that if the challenge second round sender OT messages contain $((x_{2,0}, \omega_0), (x_{2,1}, \omega_1))$ and $(x_{2,0,0}, x_{2,0,1})$ as the sender's input messages, then the view of $\mathcal{A}$ and the outputs of the honest parties is identical to the output of Hybrid_2. Else, it is identically distributed to the output of Hybrid_3. Thus, if $\mathcal{A}$ can distinguish between Hybrid_2 and Hybrid_3 with non-negligible advantage then, $\mathcal{B}$ breaks the sender security of two-round oblivious transfer with the same advantage. This is a contradiction. □

Claim 4.7. *Assuming the receiver security of the two-round oblivious transfer, we have* $\mathsf{Hybrid}_3 \overset{c}{\approx} \mathsf{Hybrid}_4$.

Proof. Assume for the sake of contradiction that the adversary $\mathcal{A}$ can distinguish between the outputs of Hybrid_3 and Hybrid_4 with non-negligible advantage. We will use this adversary to construct an adversary $\mathcal{B}$ that breaks the receiver security of oblivious transfer.

$\mathcal{B}$ interacts with the challenger against the receiver OT security by giving x_{2,x_1} and $x_{2,1-x_1}$ as the challenge bits. It receives otr^* from the external challenger. It generates $\mathsf{otr}_{x_1} := \mathsf{OT}_1(1^\lambda, x_{2,x_1}; \omega_{x_1})$ as in the previous hybrid and sets $\mathsf{otr}_{1-x_1} = \mathsf{otr}^*$. It broadcasts $(\mathsf{otr}_0, \mathsf{otr}_1)$ as the first round message from honest P_2. In the second round, $\mathcal{B}$ generates ots_3 as in the previous hybrid and uses $(x_{2,x_1}, \omega_{x_1})$ to generate $\mathsf{ots} := \mathsf{OT}_2(\mathsf{otr}, (x_{2,x_1}, \omega_{x_1}), (x_{2,x_1}, \omega_{x_1}))$. The rest of the steps are identical to the previous hybrid.

Notice that if otr^* contains the choice bit $x_{2,1-x_1}$ then the view of $\mathcal{A}$ along with the outputs of the honest parties is identical to Hybrid_3. Else, it is identically distributed to Hybrid_4. Thus, if $\mathcal{A}$ can distinguish between Hybrid_3 and Hybrid_4 with non-negligible advantage, then $\mathcal{B}$ can break the receiver security of the two-round oblivious transfer with non-negligible advantage which is a contradiction. □

Claim 4.8. *Assuming the sender security of two-round oblivious transfer, we have* $\mathsf{Hybrid}_5 \overset{c}{\approx} \mathsf{Hybrid}_6$.

Proof. We prove this claim by making use of an intermediate hybrid.

$\underline{\mathsf{Hybrid}'_5}$: In this hybrid, we compute $\mathsf{ots}_b \leftarrow \mathsf{OT}_2(\mathsf{otr}_b, x_{3,x_{2,b}}, x_{3,x_{2,b}})$ instead of $\mathsf{OT}_2(\mathsf{otr}_b, x_{3,0}, x_{3,1})$ for each $b \in \{0, 1\}$. We compute $\overline{\mathsf{ots}}$ as in Hybrid_5.

Notice that the only difference between Hybrid_5 and Hybrid'_5 is how ots_b is computed for each $b \in \{0,1\}$. We can use an identical argument as in Claim 4.6 and rely on the sender security of the two-round oblivious transfer to show that Hybrid_5 and Hybrid'_5 are computationally indistinguishable.

Further, we observe that the only difference between Hybrid'_5 and Hybrid_6 is in the computation of $\overline{\mathsf{ots}}$. In Hybrid_6, it is computed as $\overline{\mathsf{ots}}_1 \leftarrow \mathsf{OT}_2(\mathsf{otr}, \mathsf{ots}_{x_1}, \mathsf{ots}_{x_1})$ whereas in Hybrid'_5, it is computed as $\overline{\mathsf{ots}}_1 \leftarrow \mathsf{OT}_2(\mathsf{otr}, \mathsf{ots}_0, \mathsf{ots}_1)$. Again, via an identical argument to Claim 4.6, we can rely on the sender security of two-round oblivious transfer and show that Hybrid'_5 and Hybrid_6 are computationally indistinguishable.

This shows that Hybrid_5 and Hybrid_6 are computationally indistinguishable. □

Claim 4.9. $\mathsf{Hybrid}_7 \equiv \mathsf{Hybrid}_8$.

Proof. The proof follows from the observation that in both hybrids $z_1 + z_2 + z_3 = z$. □

Claim 4.10. $\mathsf{Hybrid}_8 \equiv \mathsf{Hybrid}_9$

Proof. The proof follows from the fact that $(\delta_1, \delta_2, \delta_3)$ form a random secret sharing of 0. □

Claim 4.11. *Assuming the sender security of two-round oblivious transfer,* $\mathsf{Hybrid}_9 \overset{c}{\approx} \mathsf{Hybrid}_{10}$.

Proof. The proof is identical to the proof of Claim 4.6. □

Claim 4.12. *Assuming the sender security of the two-round oblivious transfer, we have* $\mathsf{Hybrid}_{10} \overset{c}{\approx} \mathsf{Hybrid}_{11}$.

Proof. The proof of this claim is identical to Claim 4.8. □

Claim 4.13. $\mathsf{Hybrid}_{11} \equiv \mathsf{Hybrid}_{12}$

Proof. Recall that $(x_{2,0}, x_{2,1})$, $(x_{2,0,0}, x_{2,0,1})$ and $(x_{3,0}, x_{3,1})$ form a random secret sharing of x_2, $x_{2,0}$ and x_3 respectively. Thus, it now follows that if $P_2 \in H$ then $x_{2,x_1}, x_{2,0,x_3}$ are identically distributed to random bits. A similar argument shows that if $P_3 \in H$, then $x_{3,x_{2,x_3}}$ is identically distributed to a random bit. □

This completes the proof of security. □

4.2 Step-2: Protocol for Arbitrary Functions

We first recall the theorem about completeness of $\mathcal{F}_{\mathsf{3MULTPlus}}$ from [ABG+20, Theorem 6.4].

Theorem 4.14 ([BGI+18, GIS18, ABG+20])**.** *Let f be an n-party functionality. There exists a protocol Π_f for securely computing f against a semi-honest adversary (corrupting an arbitrary subset of parties), where Π_f makes parallel calls to the $\mathcal{F}_{\mathsf{3MULTPlus}}$ functionality and uses no further interaction. The protocol Π_f can either be: (1) computationally secure using a black-box PRG, where the complexity of the parties is polynomial in n, the security parameter λ and the circuit size of f, or alternatively (2) perfectly secure, where the complexity of the parties is polynomial in n and the branching program size of f.*

From Theorem 4.1 and the UC composition theorem [Can01], we get the following corollary.

Corollary 4.15. *Let f be an n-party functionality. There is a three-round protocol that makes black-box use of a two-round, semi-honest secure oblivious transfer and securely computes f against a semi-honest adversary corrupting an arbitrary subset of parties. The complexity of the parties is polynomial in n, the security parameter λ and the circuit size of f.*

5 3-round Malicious MPC

In this section, we give a construction of a 3-round protocol that computes any multiparty functionality with UC-security against malicious adversaries. The protocol makes black-box use of a two-round, malicious-secure oblivious transfer with equivocal receiver security. We do this in three steps. In the first step, we define a special n-party functionality called double selection and give a two-round, black-box protocol that securely computes this functionality. However, this protocol satisfies only a weaker notion of security which is security with input dependent abort (see Appendix A.6). In the second step, we use the protocol from the first step and give a three-round protocol that securely computes this double selection functionality with standard security. In the final step, we show how to bootstrap the protocol from the second step to a black-box, three-round protocol for general functions.

5.1 First Step: Special Functionality with Input Dependent Abort

In this subsection, we define a special n-party functionality $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ in Figure 2 and give a black-box, two-round protocol that computes $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$. This functionality captures input-dependent abort attack that can be launched by a corrupt P_2 against P_1, causing loss of input privacy of P_1.

Functionality $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$

$\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ is parameterized by an n-party function $\mathsf{dSelPri}$ whose description follows. $\mathsf{dSelPri}$ receives $(\alpha, r) \in \{0,1\} \times \{0,1\}$ from P_1, $(y_0, y_1) \in \{0,1\} \times \{0,1\}$ from P_2 and for every $3 \le i \le n$, it receives $(z^i_0, z^i_1) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$ from P_i. $\mathsf{dSelPri}$ delivers $(y_\alpha, \{z^i_{y_\alpha \oplus r}\}_{3 \le i \le n})$ to P_1 and the other parties do not get any outputs. Let x_i be the input of party P_i to $\mathsf{dSelPri}$ (note that x_i for different parties maybe of different lengths) and let $\mathcal{S}$ be the adversary. The functionality $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ proceeds as follows:

1. Each party P_i (and $\mathcal{S}$ on behalf of P_i if P_i is corrupted) sends $(\mathsf{input}, \mathsf{sid}, P_i, x_i)$ to the functionality.
2. If P_2 is corrupted then $\mathcal{S}$ may send $(\mathsf{predicate}, \mathsf{sid}, \mathsf{EQ}_\beta)$ where EQ_β is the equality predicate that takes the first component of P_1's input α and outputs 1 iff $\beta = \alpha$.
3. Upon receiving the inputs from all parties, evaluate $\mathsf{out} := \mathsf{dSelPri}(x_1, \ldots, x_n)$. If P_1 is corrupted, the functionality delivers out to $\mathcal{S}$.
4. If P_1 is not corrupted, then on receiving $(\mathsf{generateOutput}, \mathsf{sid})$ from $\mathcal{S}$, the ideal functionality computes $\mathsf{pred} = \mathsf{EQ}_\beta(\alpha)$ (if $(\mathsf{predicate}, \mathsf{sid}, \mathsf{EQ}_\beta)$ is received; if such a message is not received, it sets $\mathsf{pred} = 0$). If $\mathsf{pred} = 0$, it gives $(\mathsf{output}, \mathsf{sid}, P_1, \mathsf{out})$ to P_1. Else, if $\mathsf{pred} = 1$, it sends $(\mathsf{output}, \mathsf{sid}, P_1, \mathsf{abort})$. (And ignores the message if inputs from all parties in $\{P_1, \ldots, P_n\}$ have not been received.) On the other hand, if $(\mathsf{abort}, \mathsf{sid})$ is received then, it sends $(\mathsf{output}, \mathsf{sid}, P_1, \mathsf{abort})$ to P_1.

Figure 2: Functionality $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$

We show the following theorem.

Theorem 5.1. *There exists a two-round protocol* $\Pi^{\dagger}_{\mathsf{dSelPri}}$ *that UC-realizes the functionality* $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ *in the* $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ *(described in Figure 3) hybrid model making black-box access to a two-round, malicious-secure oblivious transfer with equivocal receiver security.*

As a corollary of Theorem 3.5, we get:

Corollary 5.2. *There exists a two-round protocol* $\Pi^{\dagger}_{\mathsf{dSelPri}}$ *that UC-realizes the functionality* $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ *making black-box access to a two-round, malicious-secure oblivious transfer with equivocal receiver security.*

Functionality $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$

Let $\mathcal{S}$ be an adversary.

- A party P_i (and $\mathcal{S}$ on behalf of P_i if P_i is corrupted) sends $(\mathsf{receiver}, \mathsf{sid}, P_i)$.
- Another party P_j (and $\mathcal{S}$ on behalf of P_j if P_j is corrupted) sends $(\mathsf{sender}, \mathsf{sid}, P_j, (s_1, \ldots, s_m))$ to the functionality where $s_j \in \{0,1\}^*$ for each $j \in [m]$.
- On receiving both these messages, for each $j \in [m]$, the functionality independently sets $s'_j = s_j$ with probability p and sets $s'_j = \bot$ with probability $1-p$.
- On receiving $(\mathsf{generateOutput}, \mathsf{sid})$ from $\mathcal{S}$ (if P_j is corrupted), the functionality delivers $(\mathsf{output}, \mathsf{sid}, (s'_1, \ldots, s'_m))$ to P_i.

Figure 3: Functionality $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$

Building $\Pi^{\dagger}_{\mathsf{dSelPri}}$**.** The formal description of $\Pi^{\dagger}_{\mathsf{dSelPri}}$ is given in Figure 4. We present a high-level idea first. We begin with the description of a protocol that computes a simplified version of the function $\mathsf{dSelPri}$ in the face of a semi-honest adversary. The simplified version does not involve a mask bit r from P_1 and additionally, assumes that P_3 as the lone provider of a pair z_0, z_1. The goal is to let P_1 learn (y_α, z_{y_α}) at the end of the second round. To construct a two-round protocol for the simplified functionality, we use the idea of "cascading oblivious transfer." P_1 produces an OT receiver message otr with α as the choice bit. P_2 produces two OT receiver messages $\{\mathsf{otr}_b\}_{b\in\{0,1\}}$ where otr_b is generated using y_b as the choice bit for each $b \in \{0,1\}$. For each $b \in \{0,1\}$, P_3 generates OT sender messages ots_b in response to otr_b with (z_0, z_1) as input. P_3 finally uses $\mathsf{ots}_0, \mathsf{ots}_1$ to compute an OT sender message in response to otr. This enables P_1 to obtain ots_α from P_3. Lastly, to enable P_1 to decrypt ots_α, P_2 sends a sender message to P_1 in response to otr with $((y_0, \mu_0), (y_1, \mu_1))$ as the inputs, where μ_b denotes the second component of the output of OT_1 while generating otr_b. Now, P_1 has (y_α, μ_α) and ots_α and can extract z_{y_α}.

The inclusion of the random mask bit r requires P_1 to produce another OT receiver message $\overline{\mathsf{otr}}$ with r as the input. To this, P_3 generates a sender OT message with input $(0, z_1 - z_0)$. This ensures that P_1 obtains its outcome as $z_{y_\alpha} + r(z_1 - z_0) = (y_\alpha + r)(z_1 - z_0) + z_0$ which is same as $z_{y_\alpha + r}$. Lastly, to ensure that a corrupt P_1 learns $z_{y_\alpha+r}$ and nothing beyond, P_3 chooses a random mask and uses $(z_0 + \mathsf{mask}, z_1 + \mathsf{mask})$, instead of (z_0, z_1), as the input for preparing $\mathsf{ots}_0, \mathsf{ots}_1$ and likewise, it uses $(-\mathsf{mask}, z_1 - z_0 - \mathsf{mask})$ instead of $(0, z_1 - z_0)$, as the sender input. This not only ensures that the end result remains unaffected, but also guarantees that nothing beyond the end result is learnt from the two summands.

To make the above idea work against a malicious adversary, we inspect the roles of the various parties and try to see the kind of attack that they can mount. P_1's role only includes preparing two OT receiver messages and therefore a corrupt P_1 is taken care by the sender security of the OT against malicious receivers. Next, a corrupt P_2 plays the role of two receivers to P_3 and one sender to P_1, where the messages and matching randomnesses used for the former role are fed as input in the latter role. While OT's sender security takes care, and in effect, fixes P_2's input through the receiver messages, there is still a scope for P_2 to launch a selective failure or input-dependent attack against P_1 by selectively choosing only one of the OT sender inputs correctly. This allows it to learn P_1's input α, by simply observing whether P_1 aborts or not. But the functionality $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ allows this attack, and preventing this attack is taken care in the next section. This brings us to the last case where P_3 can be corrupt.

P_3 prepares three OT sender messages, wherein the third instance takes the result of first two instances as input and in addition, the inputs to the first two instances need to be identical, namely $(z_0 + \mathsf{mask}, z_1 + \mathsf{mask})$. Tackling a corrupt P_3 clearly requires to step beyond OT receiver security against malicious senders. Here, we deploy MPC-in-the-head approach [IKOS07] for the consistency check, where P_3 prepares states of m virtual parties in its head that jointly hold a secret sharing of z_0, z_1, mask. The sharing is pairwise checkable and adheres to a threshold that dictates its security. A bivariate polynomial based sharing scheme fits the bill. Next, the i-th virtual party's state includes the OT sender messages that are prepared by simply replicating P_3's computation on the i-th shares of z_0, z_1, mask. Now, the goal is to open some number of the states to P_1 for checking and we need to ensure that this number (a) is not big enough to violate P_3's privacy, (b) but is enough to either catch a corrupt P_3 or error-correct the faults. Here, we invoke a 2-party NISC between P_1 and P_3 for computing the Rabin OT functionality $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$, where P_3 inputs the m states. $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ ensures each state is chosen to be revealed to P_1 independently with probability p. Using Chernoff bounds, we can conclude that the probability that more than the threshold number of states are revealed to P_1 is negligible. Consequently, the secrets z_0, z_1, mask are safe from P_1 with overwhelming probability. On the other hand, a corrupt P_3 either gets caught with overwhelming probability when it prepares a "large" number of wrong states and in the case where it ends up maligning small number of states, we rely on error correction to ensure the recovery of information. Since the NISC realizing $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ makes black-box use of a two-round oblivious transfer [IPS08, IKO+11], our final construction is black-box, as desired.

Protocol $\Pi^{\dagger}_{\mathsf{dSelPri}}$

Inputs: P_1 inputs $(\alpha, r) \in \{0,1\} \times \{0,1\}$, P_2 inputs $(y_0, y_1) \in \{0,1\} \times \{0,1\}$. For every $3 \leq i \leq n$, P_i inputs $(z_0^i, z_1^i) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$.

Output: P_1 outputs $(y_\alpha, \{z^i_{y_\alpha \oplus r}\}_{3 \leq i \leq n})$ and the other parties do not get any outputs.

Primitives: (a) A malicious-secure two-round OT with equivocal receiver security defined by $(K_{\mathsf{OT}}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ (see Section 3.3). We use OT_1^* to denote an algorithm that takes a crs and $q(\lambda)$-bit string (for some polynomial $q(\cdot)$) as input and applies OT_1 to each bit of that string. (b) Functionality $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ where $m = 3\lambda + 1$ and $p = \lambda/2m$.

Common Random/Reference String Generation: For each $i \in [n]$, sample $\mathsf{crs}^i \leftarrow K_{\mathsf{OT}}(1^\lambda)$. Set the crs to be $(\mathsf{crs}^1, \ldots, \mathsf{crs}^n)$.

Round-1: In the first round,

- P_1 computes $(\mathsf{otr}, \mu) \leftarrow \mathsf{OT}_1(\mathsf{crs}^1, \alpha)$ and $(\overline{\mathsf{otr}}, \overline{\mu}) \leftarrow \mathsf{OT}_1(\mathsf{crs}^1, r)$. Additionally, for each $i \in [3, n]$, P_1 sends $(\mathsf{receiver}, i, P_1)$ to the $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ functionality.
- For each $b \in \{0, 1\}$, P_2 computes $(\mathsf{otr}_b, \mu_b) \leftarrow \mathsf{OT}_1(\mathsf{crs}^2, y_b)$.
- For each $i \in [3, n]$, P_i does the following:
  - It chooses $\mathsf{mask}^i \leftarrow \{0, 1\}^\lambda$ uniformly at random.
  - It chooses three random degree-λ symmetric bivariate polynomials S_0^i, S_1^i, S_2^i over $\mathsf{GF}(2^\lambda)$ such that $S_0^i(0, 0) = z_0^i$, $S_1^i(0, 0) = z_1^i$ and $S_2^i(0, 0) = \mathsf{mask}^i$.
  - For each $j \in [m]$ and for each $\gamma \in [0, 2]$, let $f_\gamma^{i,j}(x) = S_\gamma^i(x, j)$ (where we associate j with the j-th element in $\mathsf{GF}(2^\lambda)$).
  - For each $j \in [m]$ and for each $\gamma \in [0, 2]$, it computes $(\mathsf{otr}_\gamma^{i,j}, \mu_\gamma^{i,j}) := \mathsf{OT}_1^*(\mathsf{crs}^i, f_\gamma^{i,j}(x))$.
- P_1 broadcasts $(\mathsf{otr}, \overline{\mathsf{otr}})$, P_2 broadcasts $(\mathsf{otr}_0, \mathsf{otr}_1)$ and for each $i \in [3, n]$, P_i broadcasts $\{\mathsf{otr}_\gamma^{i,j}\}_{j \in [m], \gamma \in [0,2]}$ to every party.

Round-2: In the second round,

- P_2 computes $\mathsf{ots} \leftarrow \mathsf{OT}_2(\mathsf{crs}^1, \mathsf{otr}, (y_0, \mu_0), (y_1, \mu_1))$.
- For every $i \in [3, n]$, P_i does the following for each $j \in [m]$,
  - For each $b \in \{0, 1\}$, it chooses $\tau_b^{i,j} \leftarrow \{0, 1\}^*$ and computes $\mathsf{ots}_b^{i,j} := \mathsf{OT}_2(\mathsf{crs}^2, \mathsf{otr}_b, f_0^{i,j}(0) + f_2^{i,j}(0), f_1^{i,j}(0) + f_2^{i,j}(0); \tau_b^{i,j})$.
  - It chooses random $\tau^{i,j} \leftarrow \{0, 1\}^*$ and computes $\mathsf{ots}^{i,j} := \mathsf{OT}_2(\mathsf{crs}^1, \mathsf{otr}, \mathsf{ots}_0^{i,j}, \mathsf{ots}_1^{i,j}; \tau^{i,j})$.
  - It chooses random $\overline{\tau}^{i,j} \leftarrow \{0, 1\}^*$ and computes $\overline{\mathsf{ots}}^{i,j} \leftarrow \mathsf{OT}_2(\mathsf{crs}^1, \overline{\mathsf{otr}}, -f_2^{i,j}(0), f_1^{i,j}(0) - f_0^{i,j}(0) - f_2^{i,j}(0); \overline{\tau}^{i,j})$.
  - It sets the string $s^{i,j} = (\{f_\gamma^{i,j}(x), \mu_\gamma^{i,j}\}_{\gamma \in [0,2]}, \{\mathsf{ots}_b^{i,j}, \tau_b^{i,j}\}_{b \in \{0,1\}}, \tau^{i,j}, \overline{\tau}^{i,j})$.

  It then sends $(\mathsf{sender}, i, P_i, (s^{i,1}, \ldots, s^{i,m}))$ to the $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ functionality.
- P_2 sends ots and for every $i \in [3, n]$, P_i sends $(\{\mathsf{ots}^{i,j}, \overline{\mathsf{ots}}^{i,j}\}_{j \in [m]})$ to P_1 via private channels (which can implemented in two rounds over a public-channel model using a two-round OT).

Output: To compute the output, P_1 does the following:

- For each $i \in [3, n]$,
  - It receives $(\mathsf{output}, i, (\overline{s}^{i,1}, \ldots, \overline{s}^{i,m}))$ as the output from $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ functionality.
  - Let $J_i \subseteq [m]$ such that for each $j \in J_i$, $\overline{s}_j^i \neq \perp$.
  - For each $j \in J_i$:
    * It parses $\overline{s}^{i,j}$ as $(\{f_\gamma^{i,j}(x), \mu_\gamma^{i,j}\}_{\gamma \in [0,2]}, \{\mathsf{ots}_b^{i,j}, \tau_b^{i,j}\}_{b \in \{0,1\}}, \tau^{i,j}, \overline{\tau}^{i,j})$.
    * For each $\gamma \in [0, 2]$, it checks if $\mathsf{CheckValid}(\mathsf{crs}^i, \mathsf{otr}_\gamma^{i,j}, (f_\gamma^{i,j}(x), \mu_\gamma^{i,j}))$ (where $\mathsf{CheckValid}$ is the algorithm for checking the validity of receiver's key (see Section 3.3)) outputs 1 and if $f_\gamma^{i,j}(x)$ is a degree-λ polynomial.
    * For every $k \in J_i \setminus \{j\}$ and $\gamma \in [0, 2]$, it checks if $f_\gamma^{i,j}(k) = f_\gamma^{i,k}(j)$.
    * It checks if $\mathsf{ots}^{i,j} := \mathsf{OT}_2(\mathsf{crs}^1, \mathsf{otr}, \mathsf{ots}_0^{i,j}, \mathsf{ots}_1^{i,j}; \tau^{i,j})$ and $\overline{\mathsf{ots}}^{i,j} \leftarrow \mathsf{OT}_2(\mathsf{crs}^1, \overline{\mathsf{otr}}, -f_2^{i,j}(0), f_1^{i,j}(0) - f_0^{i,j}(0) - f_2^{i,j}(0); \overline{\tau}^{i,j})$.
    * It also checks if $\mathsf{ots}_b^{i,j} := \mathsf{OT}_2(\mathsf{crs}^2, \mathsf{otr}_b, f_0^{i,j}(0) + f_2^{i,j}(0), f_1^{i,j}(0) + f_2^{i,j}(0); \tau_b^{i,j})$ for each $b \in \{0, 1\}$.
    * If any of the above checks fail, it aborts.

- It computes $(y_\alpha, \mu_\alpha) := \mathsf{OT}_3(\mathsf{crs}^1, \mathsf{ots}, (\alpha, \mu))$. It then runs $\mathsf{CheckValid}(\mathsf{crs}^2, \mathsf{otr}_\alpha, (y_\alpha, \mu_\alpha))$. If the algorithm outputs 1, then it proceeds. Otherwise, it aborts.
- For each $j \in [m]$,
  * It computes $\mathsf{ots}_\alpha^{i,j} := \mathsf{OT}_3(\mathsf{crs}^1, \mathsf{ots}^{i,j}, (\alpha, \mu))$.
  * It then computes $\mathsf{Sh}_{y_\alpha}^{i,j} := \mathsf{OT}_3(\mathsf{crs}^2, \mathsf{ots}_\alpha^{i,j}, (y_\alpha, \mu_\alpha))$.
  * It also computes $\overline{\mathsf{Sh}}_r^{i,j} := \mathsf{OT}_3(\mathsf{crs}^1, \overline{\mathsf{ots}}^{i,j}, (r, \overline{\mu}))$.
- It computes z_i as the Reed-Solomon decoding of $\{\mathsf{Sh}_{y_\alpha}^{i,j} + \overline{\mathsf{Sh}}_r^{i,j}\}_{j \in [m]}$, correcting at most λ errors.

It outputs $(y_\alpha, \{z_i\}_{i \in [3,n]})$.

Figure 4: Protocol $\Pi^\dagger_{\mathsf{dSelPri}}$

The following lemma is sufficient to prove Theorem 5.1.

Lemma 5.3. *Let $\mathcal{A}$ be an (possibly malicious) adversary corrupting an arbitrary subset of parties in the protocol $\Pi^\dagger_{\mathsf{dSelPri}}$. There exists a simulator Sim such that for any environment $\mathcal{Z}$,*

$$\mathrm{EXEC}_{\mathcal{F}^\dagger_{\mathsf{dSelPri}}, \mathsf{Sim}, \mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}_{\Pi^\dagger_{\mathsf{dSelPri}}, \mathcal{A}, \mathcal{Z}}$$

Proof. Let $C \subset [n]$ be the set of parties corrupted by $\mathcal{A}$ and let $H = \{P_1, \ldots, P_n\} \setminus C$ denote the set of uncorrupted parties. Since we assume that $\mathcal{A}$ is static, the set of corrupted parties C is decided before the beginning of the protocol. We now give the description of the ideal world simulator Sim. Sim internally uses the simulators $(\mathsf{Sim}_R, \mathsf{Sim}_S)$ of the oblivious transfer (see Section 3.3).

Interaction with environment $\mathcal{Z}$. For every input value corresponding to the set of corrupted parties C that Sim receives from the environment $\mathcal{Z}$, Sim writes this value to $\mathcal{A}$'s input tape. Similarly, the contents of $\mathcal{A}$'s output tape is written to Sim's output tape. We now describe how Sim simulates the interaction of honest parties with $\mathcal{A}$.

Simulating the interaction with $\mathcal{A}$: For every concurrent interaction with the session identifier sid that $\mathcal{A}$ may start, the simulator does the following:

Common Random/Reference String Generation: Sim generates the crs as follows:

- For each $i \in [n]$, if $P_i \in C$, then Sim samples $\underline{(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}_S^1(1^\lambda)}$. Else, it samples $\underline{(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}_R^1(1^\lambda)}$.
- Sim sets the crs to be $(\mathsf{crs}^1, \ldots, \mathsf{crs}^n)$.

Round-1 messages from Sim to $\mathcal{A}$: Sim does the following:

- If $P_1 \in H$, it samples $\underline{(\mathsf{otr}, \mu_0, \mu_1) \leftarrow \mathsf{Sim}_R^2(\mathsf{crs}^1, \mathsf{td}^1)}$ and $\underline{(\overline{\mathsf{otr}}, \overline{\mu}_0, \overline{\mu}_1) \leftarrow \mathsf{Sim}_R^2(\mathsf{crs}^1, \mathsf{td}^1)}$.
- If $P_2 \in H$, then for each $b \in \{0, 1\}$, it samples $\underline{(\mathsf{otr}_b, \mu_{b,0}, \mu_{b,1}) \leftarrow \mathsf{Sim}_R^2(\mathsf{crs}^2, \mathsf{td}^2)}$.
- For each $i \in [3, n]$, if $P_i \in H$, it does the following:

– For each $j \in [m]$, it independently sets $\mathsf{coin}^{i,j} = 1$ with probability p and $\mathsf{coin}^{i,j} = 0$ with probability $1 - p$.
– If $\sum_{j \in [m]} \mathsf{coin}^{i,j} > \lambda$, it aborts and outputs a special symbol error.
– Else, it samples random symmetric bivariate polynomials S_0^i, S_1^i, S_2^i with degree-λ over $\mathsf{GF}(2^\lambda)$.
– For each $j \in [m]$ and for each $\gamma \in [0, 2]$, let $f_\gamma^{i,j}(x) = S_\gamma^i(x, j)$ (where we associate j with the j-th element in $\mathsf{GF}(2^\lambda)$).
– For each $j \in [m]$ such that $\mathsf{coin}^{i,j} = 1$ and for each $\gamma \in [0, 2]$, it computes $(\mathsf{otr}_\gamma^{i,j}, \mu_\gamma^{i,j}) := \mathsf{OT}_1^*(\mathsf{crs}^i, f_\gamma^{i,j}(x))$.
– For each $j \in [m]$ such that $\mathsf{coin}^{i,j} = 0$, it computes $(\mathsf{otm}_\gamma^{i,j}, \mu_\gamma^{i,j}) \leftarrow \mathsf{OT}_1^*(\mathsf{crs}^i, 0^{2\lambda(\lambda+1)})$.

- It sends the first round messages on behalf of the honest parties to $\mathcal{A}$.

Round-2 messages from Sim to $\mathcal{A}$: To generate the round-2 messages, Sim does the following:

- If $P_1 \in C$, it runs $\mathsf{Sim}_S^2(\mathsf{crs}^1, \mathsf{td}^1, \mathsf{otr})$ to obtain α and $\mathsf{Sim}_S^2(\mathsf{crs}^1, \mathsf{td}^1, \overline{\mathsf{otr}})$ to obtain r. It sets $x_1 = (\alpha, r)$.

- If $P_2 \in C$, for each $b \in \{0, 1\}$, it runs $\mathsf{Sim}_S^2(\mathsf{crs}^2, \mathsf{td}^2, \mathsf{otr}_b)$ to obtain y_b. It sets $x_2 = (y_0, y_1)$

- For each $i \in [3, n]$, if $P_i \in C$, for each $\gamma \in [0, 2]$ and for each $j \in [m]$, it runs $\mathsf{Sim}_S^2(\mathsf{crs}^i, \mathsf{td}^i, \mathsf{otr}_\gamma^{i,j})$ to recover $f_\gamma^{i,j}(x)$. It applies the Reed-Solomon decoding on $\{f_0^{i,j}(0)\}_{j \in [m]}$ and $\{f_1^{i,j}(0)\}_{j \in [m]}$ to obtain z_0^i and z_1^i respectively. It sets $x_i = (z_0^i, z_1^i)$.

- For each $i \in [n]$ such that $P_i \in C$, it sends $(\mathsf{input}, \mathsf{sid}, P_i, x_i)$ to the ideal functionality.

- If $P_1 \in C$, Sim does the following:
  – It obtains the output $(y_\alpha, \{z_{y_\alpha \oplus r}^i\}_{i \in [3,n]})$ from the ideal functionality.
  – If $P_2 \in H$, then it computes $\mathsf{ots} \leftarrow \mathsf{OT}_2(\mathsf{crs}^1, \mathsf{otr}, (y_\alpha, \mu_{y_\alpha}^\alpha), (y_\alpha, \mu_{y_\alpha}^\alpha))$.
  – For each $i \in [3, n]$, if $P_i \in H$, Sim does the following:
    * It chooses a random mask $\mathsf{mask}^i \leftarrow \{0, 1\}^\lambda$.
    * Let $J_i \subseteq [m]$ such that $\mathsf{coin}^{i,j} = 1$.
    * It resamples S_1^i, S_2^i such that for each $j \in J_i$ and $\gamma \in [1, 2]$, $S_\gamma^i(x, j) = f_\gamma^{i,j}(x)$, $S_1^i(0, 0) = z_{y_\alpha \oplus r}^i$, and $S_2^i(0, 0) = \mathsf{mask}^i$.
    * For every $j \in [m] \setminus J_i$ and $\gamma \in [1, 2]$, it resets $f_\gamma^{i,j}(x) = S_\gamma^i(x, j)$.
    * For each $j \in [m] \setminus J_i$, it chooses $\tau_\alpha^{i,j} \leftarrow \{0, 1\}^*$ and computes $\mathsf{ots}_\alpha^{i,j} := \mathsf{OT}_2(\mathsf{crs}^2, \mathsf{otr}_\alpha, f_1^{i,j}(0) + f_2^{i,j}(0), f_1^{i,j}(0) + f_2^{i,j}(0); \tau_b^{i,j})$. For each $j \in J_i$, it computes $\{\mathsf{ots}_b^{i,j}\}_{b \in \{0,1\}}$ as in the original protocol.
    * It chooses random $\tau^{i,j} \leftarrow \{0, 1\}^*$ and computes $\mathsf{ots}^{i,j} := \mathsf{OT}_2(\mathsf{crs}^1, \mathsf{otr}, \mathsf{ots}_\alpha^{i,j}, \mathsf{ots}_\alpha^{i,j}; \tau^{i,j})$ for each $j \in [m] \setminus J_i$. For each $j \in J_i$, it generates $\mathsf{ots}^{i,j}$ as in the original protocol.

    * It chooses random $\overline{\tau}^{i,j} \leftarrow \{0,1\}^*$ and computes $\overline{\mathsf{ots}}^{i,j} \leftarrow \mathsf{OT}_2(\mathsf{crs}^1, \overline{\mathsf{otr}}, -f_2^{i,j}(0), -f_2^{i,j}(0); \overline{\tau}^{i,j})$ for each $j \in [m] \setminus J_i$. For each $j \in J_i$, it computes $\overline{\mathsf{ots}}^{i,j}$ as in the original protocol.
    * For each $j \in J_i$, it sets the string $s^{i,j} = (\{f_\gamma^{i,j}(x), \mu_\gamma^{i,j}\}_{\gamma\in[0,2]}, \{\mathsf{ots}_b^{i,j}, \tau_b^{i,j}\}_{b\in\{0,1\}}, \tau^{i,j}, \overline{\tau}^{i,j})$ and for $j \in [m] \setminus J_i$, it sets $s^{i,j} = \perp$.
    * It gives $(\mathsf{output}, i, (s^{i,1}, \dots, s^{i,m}))$ as the output from $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ functionality to P_1.

- If $P_1 \in H$, since all the messages in the second round are sent to P_1 via private channels, Sim does not have to simulate any second round messages from honest parties.

- Sim sends the messages generated on behalf of the honest parties to $\mathcal{A}$.

Output phase: If $P_1 \in H$, then Sim does the following.

- For each $i \in [3, n]$ such that $P_i \in C$,
  - It intercepts the message $(\mathsf{sender}, i, P_i, (s^{i,1}, \dots, s^{i,m}))$ that P_i sends to the $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ functionality.
  - It initializes a graph G with the vertex set to be $[m]$ and no edges.
  - For each $j \in [m]$,
    * It parses $s^{i,j}$ as $(\{f_\gamma^{i,j}(x), \mu_\gamma^{i,j}\}_{\gamma\in[0,2]}, \{\mathsf{ots}_b^{i,j}, \tau_b^{i,j}\}_{b\in\{0,1\}}, \tau^{i,j}, \overline{\tau}^{i,j})$.
    * For each $\gamma \in [0, 2]$, it checks if $\mathsf{CheckValid}(\mathsf{crs}^i, \mathsf{otr}_\gamma^{i,j}, (f_\gamma^{i,j}(x), \mu_\gamma^{i,j}))$ outputs 1 and if $f_\gamma^{i,j}(x)$ is a degree-λ polynomial. If not, it adds an edge from j to every other vertex in the graph G.
    * For every $k \in [m] \setminus \{j\}$ and $\gamma \in [0, 2]$, it checks if $f_\gamma^{i,j}(k) = f_\gamma^{i,k}(j)$. If not, it adds an edge from vertex j to vertex k in G.
    * It checks if $\mathsf{ots}^{i,j} := \mathsf{OT}_2(\mathsf{crs}^1, \mathsf{otr}, \mathsf{ots}_0^{i,j}, \mathsf{ots}_1^{i,j}; \tau^{i,j})$ and $\overline{\mathsf{ots}}^{i,j} \leftarrow \mathsf{OT}_2(\mathsf{crs}^1, \overline{\mathsf{otr}}, -f_2^{i,j}(0), f_1^{i,j}(0) - f_0^{i,j}(0) - f_2^{i,j}(0); \overline{\tau}^{i,j})$. If not, it adds an edge from j to every other vertex in the graph G.
    * It also checks if $\mathsf{ots}_b^{i,j} := \mathsf{OT}_2(\mathsf{crs}^2, \mathsf{otr}_b, f_0^{i,j}(0) + f_2^{i,j}(0), f_1^{i,j}(0) + f_2^{i,j}(0); \tau_b^{i,j})$ for each $b \in \{0,1\}$. If not, it adds an edge from j to every other vertex in the graph G.
  - It runs the 2-approximation algorithm to find the minimum vertex cover for the graph G. Let B be the vertex cover output by the algorithm.
  - If $|B| > \lambda$ for each corrupted P_i, then Sim sends $(\mathsf{abort}, \mathsf{sid})$ to the ideal functionality. If $|B| \le \lambda$, it proceeds to the next step.

- If $P_2 \in C$,
  - For each $b \in \{0,1\}$, it runs $(y_b, \mu'_b) := \mathsf{OT}_3(\mathsf{crs}^1, \mathsf{ots}, (\alpha, \mu_b))$.
  - If there exists a bit b^* such that $\mathsf{CheckValid}(\mathsf{crs}^2, \mathsf{otr}_{b^*}, (y_{b^*}, \mu_{b^*})) = 0$ but $\mathsf{CheckValid}(\mathsf{crs}^2, \mathsf{otr}_{1-b^*}, (y_{1-b^*}, \mu_{1-b^*})) = 1$, then Sim sends $(\mathsf{predicate}, \mathsf{sid}, \mathsf{EQ}_{b^*})$ to the ideal functionality.

- Sim sends $(\mathsf{generateOutput}, \mathsf{sid})$ to the ideal functionality and stops.

Proof of Indistinguishability. We now show that for any environment $\mathcal{Z}$,

$$\mathrm{EXEC}_{\mathcal{F}^{\dagger}_{\mathsf{dSelPri}},\mathsf{Sim},\mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}_{\Pi^{\dagger}_{\mathsf{dSelPri}},\mathcal{A},\mathcal{Z}}$$

We show this via a hybrid argument.

- $\underline{\mathsf{Hybrid}_0}$: This corresponds to $\mathrm{EXEC}_{\Pi^{\dagger}_{\mathsf{dSelPri}},\mathcal{A},\mathcal{Z}}$ which includes the output of the adversary and the outputs of all the honest parties.
- $\underline{\mathsf{Hybrid}_1}$: For each $i \in [3, n]$ such that $P_i \in H$, we do the following:
  - For each $j \in [m]$, we independently set $\mathsf{coin}^{i,j} = 1$ with probability p and set $\mathsf{coin}^{i,j} = 0$ with probability $1 - p$.
  - Let $J_i \subseteq [m]$ such that for each $j \in J_i$, $\mathsf{coin}^{i,j} = 1$. If $|J_i| > \lambda$, we abort and output a special symbol error.
  - When P_i sends $(\mathsf{sender}, i, P_i, (s^{i,1}, \ldots, s^{i,m}))$ to the $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ functionality, for each $j \in [m]$, we set $\overline{s}^{i,j} = s^{i,j}$ if $\mathsf{coin}^{i,j} = 1$ and otherwise, set $\overline{s}^{i,j} = \perp$.
  - We deliver $(\mathsf{output}, i, (\overline{s}^{i,1}, \ldots, \overline{s}^{i,m}))$ as the output from $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ functionality to P_1.
- $\underline{\mathsf{Hybrid}_2}$: In this hybrid, we make the following changes:
  - **CRS Generation.** Instead of sampling crs^1 as the output of $K_{\mathsf{OT}}(1^\lambda)$, if $P_1 \in H$, we sample $(\mathsf{crs}^1, \mathsf{td}^1) \leftarrow \mathsf{Sim}^1_R(1^\lambda)$ and otherwise, we sample $(\mathsf{crs}^1, \mathsf{td}^1) \leftarrow \mathsf{Sim}^1_S(1^\lambda)$. We include the above sampled crs^1 as part of the crs.
  - **Round-1 message from P_1.** Skip this change if $P_1 \in C$. Instead of generating otr and $\overline{\mathsf{otr}}$ as the first component of the outputs $\mathsf{OT}_1(\mathsf{crs}^1, \alpha)$ and $\mathsf{OT}_1(\mathsf{crs}^1, r)$ respectively, we generate them as the first component of the outputs of two independent executions of $\mathsf{Sim}^2_R(\mathsf{crs}^1, \mathsf{td}^1)$.
  - Skip the following changes if $P_1 \in H$.
    - **Input Extraction.** Let $\mathsf{otr}, \overline{\mathsf{otr}}$ be the messages sent by $\mathcal{A}$ in the first round on behalf of P_1. We first run $\mathsf{Sim}^2_S(\mathsf{crs}^1, \mathsf{td}^1, \mathsf{otr})$ and $\mathsf{Sim}^2_S(\mathsf{crs}^1, \mathsf{td}^1, \overline{\mathsf{otr}})$ to obtain α and r respectively.
    - **Round-2 message from honest P_2.** If $P_2 \in H$, we generate $\mathsf{ots} \leftarrow \mathsf{OT}_2(\mathsf{crs}^1, \mathsf{otr}, (y_\alpha, \mu_\alpha), (y_\alpha, \mu_\alpha))$.
    - **Round-2 message from honest P_i where $i \in [3, n]$.** For every $i \in [3, n]$ for which $P_i \in H$ and for each $j \in [m] \setminus J_i$ (recall that J_i is the set of indices j for which $\mathsf{coin}^{i,j} = 1$), we generate $\mathsf{ots}^{i,j} \leftarrow \mathsf{OT}_2(\mathsf{crs}^1, \mathsf{otr}, \mathsf{ots}^{i,j}_\alpha, \mathsf{ots}^{i,j}_\alpha)$. Further, let $(\mathsf{msg}^{i,j}_0, \mathsf{msg}^{i,j}_1) = (-f^{i,j}_2(0), f^{i,j}_1(0) - f^{i,j}_0(0) - f^{i,j}_2(0))$. We generate $\overline{\mathsf{ots}}^{i,j} \leftarrow \mathsf{OT}_2(\mathsf{crs}^1, \overline{\mathsf{otr}}, \mathsf{msg}^{i,j}_r, \mathsf{msg}^{i,j}_r)$.
- $\underline{\mathsf{Hybrid}_3}$: In this hybrid, we make the following changes:
  - **CRS Generation.** Instead of sampling crs^2 as the output of $K_{\mathsf{OT}}(1^\lambda)$, if $P_2 \in H$, we sample $(\mathsf{crs}^2, \mathsf{td}^2) \leftarrow \mathsf{Sim}^1_R(1^\lambda)$ and otherwise, we sample $(\mathsf{crs}^2, \mathsf{td}^2) \leftarrow \mathsf{Sim}^1_S(1^\lambda)$. We include the above sampled crs^2 as part of the crs.

- **Round-1 message from P_2.** Skip this change if $P_2 \in C$. Instead of generating otm_1^0 and otm_1^1 as the first component of the outputs $\mathsf{OT}_1(\mathsf{crs}^2, y_0)$ and $\mathsf{OT}_1(\mathsf{crs}^2, y_1)$ respectively, we generate them as the first component of the outputs of two independent executions of $\mathsf{Sim}_R^2(\mathsf{crs}^2, \mathsf{td}^2)$.
- **Round-2 message from P_2.** Skip this change if $P_2 \in C$. In this hybrid, if $P_1 \in C$, we generate ots as $\mathsf{OT}_2(\mathsf{crs}^1, \mathsf{otr}, (y_\alpha, \mu^\alpha_{y_\alpha}), (y_\alpha, \mu^\alpha_{y_\alpha}))$ where for each $b \in \{0,1\}$, $(\mathsf{otr}_b, \mu_{b,0}, \mu_{b,1}) \leftarrow \mathsf{Sim}_R^2(\mathsf{crs}^2, \mathsf{td}^2)$
- Skip the following changes if $P_2 \in H$.
  - **Input Extraction.** Let $\mathsf{otr}_0, \mathsf{otr}_1$ be the first round messages that $\mathcal{A}$ sends on behalf of P_2. For each $b \in \{0,1\}$, we run $\mathsf{Sim}_S^2(\mathsf{crs}^2, \mathsf{td}^2, \mathsf{otr}_b)$ to obtain y_b.
  - **Round-2 computation of honest P_i where $i \in [3,n]$.** Skip this change if there does not exist any $i \in [3,n]$ such that $P_i \in H$. For every $j \in [m] \setminus J_i$, we compute $\mathsf{ots}_b^{i,j} := \mathsf{OT}_2(\mathsf{crs}^2, \mathsf{otr}_b, f^{i,j}_{y_b}(0) + f_2^{i,j}(0), f^{i,j}_{y_b}(0) + f_2^{i,j}(0); \tau_b^{i,j})$ (for uniformly chosen $\tau_b^{i,j}$). The rest of the computation is exactly as in the previous hybrid.

- <u>Hybrid_4</u> : In this hybrid, we make the following changes:
  - **CRS Generation.** For each $i \in [3,n]$, if $P_i \in H$, we generate $(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}_R^1(1^\lambda)$ and if $P_i \in C$, we generate $(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}_S^1(1^\lambda)$.
  - **Round-1 message from honest P_i where $i \in [3,n]$.** For every $i \in [3,n]$ such that P_i is honest and for each $j \in [m] \setminus J_i$ and for each $\gamma \in [0,2]$, generate $\mathsf{otr}_\gamma^{i,j}$ as the first component of $\mathsf{OT}_1^*(\mathsf{crs}^i, 0^{2\lambda(\lambda+1)})$.
  - Skip the following changes if $P_i \in H$.
    - **Input Extraction.** Let $\{\mathsf{otr}_\gamma^{i,j}\}_{j \in [m], \gamma \in [0,2]}$ be the first round messages sent by P_i. We run $\mathsf{Sim}_S^2(\mathsf{crs}^i, \mathsf{td}^i, \mathsf{otr}_\gamma^{i,j})$ to recover $f_\gamma^{i,j}(x)$. We then apply the Reed-Solomon decoding on $\{f_0^{i,j}(0)\}_{j \in [m]}$ and $\{f_1^{i,j}(0)\}_{j \in [m]}$ to obtain z_0^i and z_1^i respectively.

- <u>Hybrid_5</u> : Skip this hybrid if $P_1 \in H$. In this hybrid, for each $i \in [3,n]$ such that $P_i \in H$, we make the following changes in the generation of the round-1 and round-2 messages:
  - **Round-1 message:**
    - We sample random bivariate polynomials S_0^i, S_1^i, S_2^i with degree-λ over $\mathsf{GF}(2^\lambda)$.
    - For each $j \in [m]$ and for each $\gamma \in [0,2]$, let $f_\gamma^{i,j}(x) = S_\gamma^i(x, j)$.
    - For each $j \in J_i$ and for each $\gamma \in [0,2]$, wecompute $(\mathsf{otm}_\gamma^{i,j}, \mu_\gamma^{i,j}) := \mathsf{OT}_1^*(\mathsf{crs}^i, f_\gamma^{i,j}(x))$.
  - **Round-2 message:**
    - We choose a random mask $\mathsf{mask}^i \leftarrow \{0,1\}^\lambda$.
    - We resample S_1^i, S_2^i such that for each $j \in J_i$ and $\gamma \in [1,2]$, $S_\gamma^i(x,j) = f_\gamma^{i,j}(x)$, $S_1^i(0,0) = z^i_{y_\alpha \oplus r}$, and $S_2^i(0,0) = \mathsf{mask}^i$.
    - For every $j \in [m] \setminus J_i$ and $\gamma \in [1,2]$, we reset $f_\gamma^{i,j}(x) = S_\gamma^i(x,j)$.
    - For each $j \in [m] \setminus J_i$, we choose a random $\tau_b^{i,j} \leftarrow \{0,1\}^*$ and compute $\mathsf{ots}_b^{i,j} := \mathsf{OT}_2(\mathsf{crs}^2, \mathsf{otr}_b, f_1^{i,j}(0) + f_2^{i,j}(0), f_1^{i,j}(0) + f_2^{i,j}(0); \tau_b^{i,j})$.
    - We choose a random $\tau^{i,j} \leftarrow \{0,1\}^*$ and compute $\mathsf{ots}^{i,j} := \mathsf{OT}_2(\mathsf{crs}^1, \mathsf{otr}, \mathsf{ots}_\alpha^{i,j}, \mathsf{ots}_\alpha^{i,j}; \tau^{i,j})$ for each $j \in [m] \setminus J_i$.

* We then choose random $\overline{\tau}^{i,j} \leftarrow \{0,1\}^*$ and compute $\overline{\mathsf{ots}}^{i,j} \leftarrow \mathsf{OT}_2(\mathsf{crs}^1, \overline{\mathsf{otr}}, -f_2^{i,j}(0), -f_2^{i,j}(0); \overline{\tau}^{i,j})$ for each $j \in [m] \setminus J_i$.

- $\underline{\mathsf{Hybrid}_6}$: Skip this hybrid change if $P_1 \in H$. In this hybrid, instead of using the actual inputs of honest parties to compute the output of the ideal functionality, we query the ideal functionality on the inputs of the corrupted parties and obtain the output $(y_\alpha, \{z^i_{y_\alpha \oplus r}\}_{i\in[3,n]})$. We then use this output to generate the second round messages of the protocol on behalf of the honest parties. This change is only syntactic and hence, this hybrid is identical to the previous hybrid.

- $\underline{\mathsf{Hybrid}_7}$: Skip this hybrid change if $P_1 \in C$. In this hybrid, we make the following changes. For each $i \in [3, n]$ such that $P_i \in C$,
  - We intercept the message $(\mathsf{sender}, i, P_i, (s^{i,1}, \ldots, s^{i,m}))$ that P_i sends to the $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ functionality.
  - We initialize a graph G with the vertex set to be $[m]$ and no edges and perform the same checks as in the simulation to generate the output of honest P_1.

  Note that Hybrid_7 is identically distributed to $\mathrm{EXEC}_{\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}, \mathsf{Sim}, \mathcal{Z}}$.

We now show that for each $i \in [7]$, $\mathsf{Hybrid}_i \overset{c}{\approx} \mathsf{Hybrid}_{i-1}$ by giving reductions to the security of the two-round, malicious-secure oblivious transfer protocol with equivocal receiver security.

Claim 5.4. $\mathsf{Hybrid}_0 \overset{s}{\approx} \mathsf{Hybrid}_1$

Proof. We note that the only difference between Hybrid_0 and Hybrid_1 is that in some cases Hybrid_1 outputs the special symbol error and aborts. We now show that the probability that Hybrid_1 outputs error is at most $2^{-O(\lambda)}$.

$$\begin{aligned}
\Pr[\mathsf{Hybrid}_1 \text{ outputs } \mathsf{error}] &= \Pr[\exists i \in [3,n] \text{ s.t. } P_i \in H \text{ and } |J_i| > \lambda] \\
&\leq n \Pr[|J_i| > \lambda] \qquad \text{(By union bound)} \\
&= n \Pr[\sum_{j\in[m]} \mathsf{coin}^{i,j} > \lambda] \\
&\leq ne^{-\frac{\lambda}{8}} \qquad \text{(From Chernoff bounds)} \\
&\leq 2^{-O(\lambda)}
\end{aligned}$$

□

Claim 5.5. *Assume the equivocal receiver security and the sender security properties of the oblivious transfer. Then,* $\mathsf{Hybrid}_1 \overset{c}{\approx} \mathsf{Hybrid}_2$.

Proof. We consider two cases depending on whether P_1 is honest or not.

- **Case-1:** $P_1 \in H$**.** Assume for the sake of contradiction that there exists a distinguisher D that can distinguish between the outputs of Hybrid_1 and Hybrid_2 with non-negligible advantage. We will construct an adversary $\mathcal{B}$ against the equivocal receiver security property of the oblivious transfer.

$\mathcal{B}$ interacts with the challenger for the oblivious transfer protocol and gives α and r as the challenge choice bits. It receives crs^1, the first round messages $(\mathsf{otr}, \overline{\mathsf{otr}})$ and $(\mu, \overline{\mu})$. $\mathcal{B}$ uses crs^1 to generate the crs and starts the interaction with $\mathcal{A}$. It sends $(\mathsf{otr}, \overline{\mathsf{otr}})$ as the first round messages from honest P_1. It generates the rest of the first and second round messages from other honest parties as in Hybrid_1. Finally, to compute the output, it uses (α, μ) and $(r, \overline{\mu})$ as inputs to the corresponding OT_3 executions and generates the output of honest P_1 exactly as in the protocol. It finally runs the distinguisher D on the view of the adversary and the output of honest P_1 and outputs whatever D outputs.

We note that if the messages from the challenger are generated using the real algorithms then the input to D is generated identically to the output of Hybrid_1. Else, the input to D is distributed identically to the output of Hybrid_2. Hence, it follows that $\mathcal{B}$ can break the equivocal receiver security property with non-negligible advantage which is a contradiction.

- **Case-2:** $P_1 \in C$**.** Assume for the sake of contradiction that there exists a distinguisher D that can distinguish between the outputs of Hybrid_1 and Hybrid_2 with non-negligible advantage. We will construct an adversary $\mathcal{B}$ that breaks the sender security of the oblivious transfer. The description of $\mathcal{B}$ is given below.

  $\mathcal{B}$ interacts with the challenger and gives the following pairs of inputs as the sender challenge messages. If $P_2 \in H$, it sends (y_0, μ_0) and (y_1, μ_1) to the challenger. Additionally, for each $i \in [3, n]$ such that $P_i \in H$, it sends $\{\mathsf{ots}_0^{i,j}, \mathsf{ots}_1^{i,j}\}$ and $(-f_2^{i,j}(0), f_1^{i,j}(0) - f_0^{i,j}(0) - f_2^{i,j}(0))$ for each $j \in [m] \setminus J_i$ as the sender challenge messages. It receives crs^1 from the challenger and uses it to generate the crs. $\mathcal{B}$ then starts the interaction with $\mathcal{A}$. It generates the round-1 messages from the other honest parties as in Hybrid_1. It receives $(\mathsf{otr}, \overline{\mathsf{otr}})$ from $\mathcal{A}$ sent on behalf of the corrupt P_1. If $P_2 \in H$, $\mathcal{B}$ forwards otr as the first round message sent by corrupt receiver to the external challenger corresponding to the challenge messages (y_0, μ_0) and (y_1, μ_1). For each $i \in [3, n]$ such that $P_i \in H$, $\mathcal{B}$ forwards otr as the receiver message corresponding to the challenge messages $\{\mathsf{ots}_0^{i,j}, \mathsf{ots}_1^{i,j}\}$ and $\overline{\mathsf{otr}}$ as the receiver message corresponding to the challenge messages $(-f_2^{i,j}(0), f_1^{i,j}(0) - f_0^{i,j}(0) - f_2^{i,j}(0))$ for each $j \in [m] \setminus J_i$. $\mathcal{B}$ receives ots (if $P_2 \in H$) and for each $i \in [3, n]$ such that $P_i \in H$ and for each $j \in [m] \setminus J_i$, it obtains $\mathsf{ots}^{i,j}, \overline{\mathsf{ots}}^{i,j}$ from the external challenger. It generates the rest of the second-round messages sent on behalf of the honest parties as in Hybrid_1. At the end, it runs D on the view of $\mathcal{A}$ and outputs whatever it outputs.

  Note that if the view generated by the external challenger corresponds to the real world distribution then the input to D is distributed identically to the output of Hybrid_1. Else, it is distributed identically to the output of Hybrid_2. Hence, it follows that $\mathcal{B}$ can break the sender security of the oblivious transfer with non-negligible advantage which is a contradiction.

□

Claim 5.6. *Assume the equivocal receiver security and the sender security properties of the oblivious transfer. Then,* $\mathsf{Hybrid}_2 \overset{c}{\approx} \mathsf{Hybrid}_3$.

Proof. We consider two cases depending on whether P_2 is honest or not.

- **Case-1:** $P_2 \in H$**.** Assume for the sake of contradiction that there exists a distinguisher D that can distinguish between the outputs of Hybrid_2 and Hybrid_3 with non-negligible advantage. We will construct an adversary $\mathcal{B}$ against the equivocal receiver security property of the oblivious transfer.

$\mathcal{B}$ interacts with the challenger for the oblivious transfer protocol and gives y_0 and y_1 as the challenge choice bits. It receives crs^2, the first round messages $(\mathsf{otr}_0, \mathsf{otr}_1)$ and (μ_0, μ_1). $\mathcal{B}$ uses crs^2 to generate the crs and starts the interaction with $\mathcal{A}$. It sends $(\mathsf{otr}_0, \mathsf{otr}_1)$ as the first round messages from honest P_2. It generates the first round messages from the rest of the honest parties as in Hybrid_2. To generate the second round message on behalf of P_2, it uses μ_α received from the challenger. Specifically, it generates $\mathsf{ots} \leftarrow \mathsf{OT}_2(\mathsf{crs}^1, \mathsf{otr}, (y_\alpha, \mu_\alpha), (y_\alpha, \mu_\alpha))$. As in the first round, it generates the second round messages from the rest of the honest parties as in Hybrid_2. If $P_1 \in H$, it computes the output of P_1 as in Hybrid_2. It finally runs the distinguisher D on the view of the adversary and the output of P_1 (if $P_1 \in H$) and outputs whatever D outputs.

We note that if the messages from the challenger are generated using the real algorithms then the input to D is generated identically to the output of Hybrid_2. Else, the input to D is distributed identically to the output of Hybrid_3. Hence, it follows that $\mathcal{B}$ can break the equivocal receiver security property with non-negligible advantage which is a contradiction.

- **Case-2:** $P_2 \in C$**.** Assume for the sake of contradiction that there exists a distinguisher D that can distinguish between the outputs of Hybrid_2 and Hybrid_3 with non-negligible advantage. We now construct an adversary $\mathcal{B}$ that breaks the sender security of the oblivious transfer.

  $\mathcal{B}$ interacts with the challenger and gives the following pairs of inputs as the sender challenge messages. For each $i \in [3, n]$ such that $P_i \in H$ and for each $b \in \{0, 1\}$, it sends $\{f_0^{i,j}(0) + f_2^{i,j}(0), f_1^{i,j}(0) + f_2^{i,j}(0)\}$ for each $j \in [m] \setminus J_i$ as the challenge messages. It receives crs^2 from the challenger and uses it to generate the crs. $\mathcal{B}$ then starts the interaction with $\mathcal{A}$. It generates the round-1 messages from the other honest parties as in Hybrid_2. It receives $(\mathsf{otr}_0, \mathsf{otr}_1)$ from $\mathcal{A}$ sent on behalf of the corrupt P_2. For each $i \in [3, n]$ such that $P_i \in H$ and for each $b \in \{0, 1\}$, $\mathcal{B}$ forwards otr_b as the first round message sent by corrupt receiver to the external challenger. For each $i \in [3, n]$ such that $P_i \in H$ and for each $j \in [m] \setminus J_i$ and $b \in \{0, 1\}$, $\mathcal{B}$ obtains $\mathsf{ots}_b^{i,j}$ from the external challenger. It generates the rest of the second-round messages sent on behalf of the honest parties as in Hybrid_2. If $P_1 \in H$, it computes the output of P_1 as in Hybrid_2. At the end, it runs D on the view of $\mathcal{A}$ and the output of P_1 (if $P_1 \in H$) and outputs whatever it outputs.

  Note that if the view generated by the external challenger corresponds to the real world distribution then the input to D is distributed identically to the output of Hybrid_2. Else, it is distributed identically to the output of Hybrid_3. Hence, it follows that $\mathcal{B}$ can break the sender security of the oblivious transfer with non-negligible advantage which is a contradiction.

□

Claim 5.7. *Assume the equivocal receiver security and the sender security of the oblivious transfer. Then,* $\mathsf{Hybrid}_3 \overset{c}{\approx} \mathsf{Hybrid}_4$.

Proof. We consider a sequence of hybrids $\mathsf{Hybrid}_3 \equiv \mathsf{Hybrid}_{3,2}$ up to $\mathsf{Hybrid}_{3,n} \equiv \mathsf{Hybrid}_4$ where in $\mathsf{Hybrid}_{3,i}$ for $i \in [3, n]$, we change the distribution of crs^k and the first round messages generated by P_k (if $P_k \in H$) for every $k \leq i$. To prove that $\mathsf{Hybrid}_3 \overset{c}{\approx} \mathsf{Hybrid}_4$, it is sufficient to show that for every $i \in [3, n]$, $\mathsf{Hybrid}_{3,i} \overset{c}{\approx} \mathsf{Hybrid}_{3,i-1}$. We now show this by considering the cases when P_i is honest or not.

- **Case-1:** $P_i \in C$. In the case $P_i \in C$, the only change in $\mathsf{Hybrid}_{3,i}$ and in $\mathsf{Hybrid}_{3,i-1}$ is in how crs^i is generated (the input extraction step does not affect the view of the adversary). Note that in $\mathsf{Hybrid}_{3,i-1}$, crs^i is generated as the output of $K_{\mathsf{OT}}(1^\lambda)$ whereas in $\mathsf{Hybrid}_{3,i}$, it is generated as the first component of the output of $\mathsf{Sim}^1_S(1^\lambda)$. Hence, it follows directly from the sender security of oblivious transfer that $\mathsf{Hybrid}_{3,i} \overset{c}{\approx} \mathsf{Hybrid}_{3,i-1}$.

- **Case-2:** $P_i \in H$. In order to show that $\mathsf{Hybrid}_{3,i-1} \overset{c}{\approx} \mathsf{Hybrid}_{3,i}$ when $P_i \in H$, we consider an intermediate distribution $\mathsf{Hybrid}'_{3,i-1}$. In this distribution, crs^i is generated as the first component of the output of $\mathsf{Sim}^1_R(1^\lambda)$ and additionally, $\mathsf{otr}^{i,j}_\gamma$ is generated as output of $\mathsf{Sim}^2_R(\mathsf{crs}^i, \mathsf{td}^i)$ (where td^i is generated by Sim^1_R) for each $\gamma \in [0, 2]$. We can show that $\mathsf{Hybrid}_{3,i-1} \overset{c}{\approx} \mathsf{Hybrid}'_{3,i-1}$ and $\mathsf{Hybrid}'_{3,i-1} \overset{c}{\approx} \mathsf{Hybrid}_{3,i}$ via a reduction to the equivocal receiver security in a similar manner to the proofs of Claim 5.6 and Claim 5.5.

□

Claim 5.8. $\mathsf{Hybrid}_4 \overset{s}{\approx} \mathsf{Hybrid}_5$.

Proof. For any $i \in [3, n]$, consider any $P_i \in H$. We note that in Hybrid_4, the α component of the sender messages in $\{\mathsf{ots}^{i,j}\}_{j\in[m]}$ constitute a λ-out-of-m secret sharing of the field element $y_\alpha(z^i_1 - z^i_0) + z^i_0 + \mathsf{mask}^i$. Similarly, the r component of the sender messages in $\{\overline{\mathsf{ots}}^{i,j}\}_{j\in[m]}$ constitute a λ-out-of-m secret sharing of the field element $r(z^i_1 - z^i_0) - \mathsf{mask}^i$. Since $|J_i| \leq \lambda$, it follows from fact 3.8 that conditioned on the view of the adversary mask^i is uniformly distributed. Hence, Hybrid_4 is distributed identically to the distribution where α component of the sender messages in $\{\mathsf{ots}^{i,j}\}_{j\in[m]}$ constitute a λ-out-of-m secret sharing of the field element $(y_\alpha \oplus r)(z^i_1 - z^i_0) + z^i_0 + \mathsf{mask}^i$ and the r component of the sender messages in $\{\overline{\mathsf{ots}}^{i,j}\}_{j\in[m]}$ constitute a λ-out-of-m secret sharing of the field element $-\mathsf{mask}^i$. Note that this intermediate distribution is identical to Hybrid_5 from Fact 3.8 since $|J_i| \leq \lambda$. □

Claim 5.9. $\mathsf{Hybrid}_6 \overset{s}{\approx} \mathsf{Hybrid}_7$

Proof. Consider some $i \in [3, n]$ such that $P_i \in C$. We first show that if B output by the 2-approximation algorithm has size greater than λ, then an honest P_1 in Hybrid_6 also outputs abort except with probability $2^{-O(\lambda)}$.

Note that if $|B| > \lambda$, then the size of the minimum vertex cover for the graph G is of size $> \lambda/2$. This means that the maximum matching in the graph is of size $> \lambda/4$. If for at least one edge, (a, b) of this matching, both s^i_a and s^i_b are non-$\perp$, then P_1 will abort in Hybrid_6. For any edge (a, b) in the matching, the probability that either s^i_a or s^i_b is $\perp$ is at most $1 - p^2$. Since $p = \frac{\lambda}{2m}$ and $m = 3\lambda + 1$, we have $\frac{1}{7} < p < \frac{1}{6}$ and so $1 - p^2$ is at most a constant. This event is independent for each edge of the matching. Thus, for every edge of the matching, the s^i's corresponding to at least one of the vertices are set to $\perp$ by the $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ functionality is $2^{-O(\lambda)}$. Thus, P_1 in Hybrid_6 outputs abort except with probability $2^{-O(\lambda)}$.

On the other hand, if $|B| \leq \lambda$, it follows from Fact 3.7 and the error correcting properties of the Reed-Solomon codes that z^i_0, z^i_1 extracted by the simulator in Hybrid_7 is consistent with the outputs obtained by honest P_1 in Hybrid_6. Specifically, if $y_\alpha \oplus r = 0$ then honest P_1 in Hybrid_6 obtains the extracted value z_0^i (and vice versa). This completes the proof of the claim. □

□

5.2 Conforming Protocols and The Round-collapsing Compiler

The steps 2 and 3 of building a maliciously-secure MPC protocol for a general function require the usage of a conforming protocol introduced in [GS18]. In this subsection, we recall this notion and present a slightly modified version given in [GIS18]. Further, these two steps will build upon the round-collapsing compiler of [GS18] and we give an informal description in this sub-section.

Specification of a Conforming Protocol. Consider an n-party deterministic[2] MPC protocol Φ between parties $P_1, \dots, P_n$ with inputs $x_1, \dots, x_n$, respectively computing some function $f(x_1, \dots, x_n)$. For each $i \in [n]$, we let $x_i \in \{0,1\}^m$ denote the input of party P_i. A conforming protocol Φ is defined by functions pre, post, and computations steps or what we call *actions* $\phi_1, \cdots \phi_T$. The protocol Φ proceeds in three stages: the pre-processing stage, the computation stage and the output stage.

- **Pre-processing phase**: For each $i \in [n]$, party P_i first samples $v_i \in \{0,1\}^\ell$ (where ℓ is the parameter of the protocol) as the output of a randomized function $\mathsf{pre}(1^\lambda, i)$ and sets z_i as

$$z_i = (x_i \oplus v_i[(i-1)\ell/n + 1, (i-1)\ell/n + m]) \| 0^{\ell/n - m}$$

  where $v_i[(i-1)\ell/n + 1, (i-1)\ell/n + m]$ denotes the bits of the string v_i in the positions $[(i-1)\ell/n + 1, (i-1)\ell/n + m]$. P_i retains v_i as the secret information and broadcasts z_i to every other party. We require that $v_i[k] = 0$ for all $k \in [\ell] \setminus \{(i-1)\ell/n + 1, \dots, i\ell/n\}$.[3]

- **Computation phase**: For each $i \in [n]$, party P_i sets $\mathsf{st} := (z_1 \| \cdots \| z_n)$. Next, for each $t \in \{1 \cdots T\}$ parties proceed as follows:

  1. Parse action ϕ_t as (i, f, g, h) where $i \in [n]$ and $f, g, h \in [\ell]$.
  2. Party P_i computes *one* NAND gate as $\mathsf{st}[h] = \mathsf{NAND}\Big(\mathsf{st}[f] \oplus v_i[f], \mathsf{st}[g] \oplus v_i[g]\Big) \oplus v_i[h]$ and broadcasts $\mathsf{st}[h]$ to every other party.
  3. Every party P_j for $j \neq i$ updates $\mathsf{st}[h]$ to the bit value received from P_i.

  We require that for all $t, t' \in [T]$ such that $t \neq t'$, we have that if $\phi_t = (\cdot, \cdot, \cdot, h)$ and $\phi_{t'} = (\cdot, \cdot, \cdot, h')$ then $h \neq h'$. Also, we denote $A_i \subset [T]$ to be the set of rounds in with party P_i sends a bit. Namely, $A_i = \{t \in T \mid \phi_t = (i, \cdot, \cdot, \cdot)\}$.

- **Output phase**: For each $i \in [n]$, party P_i outputs $\mathsf{post}(\mathsf{st}, v_i)$.

We now recall the following theorem proved in [GS18].

[2] Randomized protocols can be handled by including the randomness used by a party as part of its input.

[3] Here, we slightly differ from the formulation used in [GS18, GIS18]. In their work, pre is defined to additionally take x_i as input and outputs (z_i, v_i). However, the transformation from any protocol to a conforming protocol given in these works has the above structure where the last $\ell/n - m$ bits of z_i are 0 and the first m bits of z_i is the XOR of x_i and $v_i[(i-1)\ell/n + 1, (i-1)\ell/n + m]$.

Theorem 5.10 ([GS18]). *Any MPC protocol* Π *can be transformed into a conforming protocol* Φ *while inheriting the correctness and the security of the original protocol. Furthermore, the* post *function of* Φ *is just a projection function (i.e., it outputs some bits of* st*)*[4] *and the simulated message* z_i *(for every honest party) is* $(r_i \| 0^{\ell/n-m})$ *where* r_i *is a uniformly chosen random string of length* m *(independent of other simulated messages).*

The High-level Idea of the Round-collapsing Compiler. We first recall the core idea used in the construction of the round-collapsing compiler of [GS18, GIS18]. In the above mentioned works, each party runs the pre-processing phase of the conforming protocol to obtain the public state z_i and the private state v_i. In simple terms, the public state of any P_i can be viewed as a large array with the initial positions filled in with the masked input of P_i and the subsequent positions left empty. Looking ahead, these empty spots would be filled with the bits that are sent and received during the computation phase of the conforming protocol. The private state, also an array, contains the masks. The positions in this array that correspond to bits that are revealed to all parties are set to 0 and the rest are populated with bits that are uniformly picked. At the end of the pre-processing step, every P_i broadcasts the public state z_i, while retaining the private state v_i. The concatenation of the public states generated by all the parties constitutes the *joint public state*. In every round of the computation phase of Φ, one position of the joint public state gets updated based on the bit that is communicated in that round (recall that in each round of Φ, only one party sends a single bit). The message bit sent by a party in each round is a function of the private state of the speaker and the joint public state updated until that round. In the two-round protocol, the round-by-round updates on the public state are emulated via a bunch of garbled circuits (precisely n garbled circuits, one for each party, corresponding to every round of Φ). All the garbled circuits that emulate a party's role in the conforming protocol has that party's private state hardwired. The garbled circuit corresponding to party P_i in the rth round takes as input the entire public state after $(r-1)$-th round and outputs the labels for the next garbled circuit that corresponds to the updated state. When the party P_i is the speaker in the r-th round, this update is local and hence, the garbled circuit can trivially output the labels corresponding to the updated state. On the other hand, if P_i is not the speaker, then the garbled circuit can only output the labels for all the positions in the state except the one that will be updated. The main technical contribution of the round-squishing technique is to use OTs to release the label for the updated position for a listening party. We now explain how this is done.

In the compiled protocol, for every bit to be communicated (which is assumed to be an output of a NAND gate), the speaker party generates 4 OT receiver messages encoding the output of the NAND gate on all four possible inputs. These messages are broadcasted to the other parties during the first round of the protocol. The actual receiver OT message which contains the correct bit to be sent in that round is determined by the updated joint public state until the previous round. Now, the speaker's garbled circuit in the r-th round not only outputs the labels corresponding to the updated state but also outputs the randomness corresponding to one of the four OT receiver message that contains the correct bit to be sent. On the other hand, the listening garbled circuit in r-th round outputs the corresponding OT sender message, prepared using the labels (of its next circuit) for the bit position to be updated in round r. This allows every party to learn the label for the communicated bit in the listening party's next garbled circuit. With the above, all that

[4] We note that this property can be generically added to any conforming protocol by expanding the computation phase to include more actions.

remains is to publish the labels for the first set of GCs corresponding the joint public state. This would trigger the execution of Φ emulated using garbled circuits. In the compiler of [GS18, GIS18] this step is done in round 2, since the public state of every party is sent in round-1 and hence, the labels for the joint state can be made available in round-2. Subsequently, the series of garbled circuit evaluation and the output computation take place locally at every party's end.

5.3 Second Step: Special Functionality with Standard Security

In this subsection, we define the n-party version of the double-selection functionality $\mathcal{F}_{\mathsf{dSel}}$ and give a three-round protocol for securely realizing this functionality. This protocol makes black-box use of a two-round malicious-secure OT with equivocal receiver security and is in the $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ hybrid model. We give the description of the function $\mathcal{F}_{\mathsf{dSel}}$ in Figure 5.

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

$\mathcal{F}_{\mathsf{dSel}}$ is parameterized by an n-party function dSel whose description follows. dSel receives $(\alpha, r) \in \{0,1\} \times \{0,1\}$ from P_1 and $(y_0, y_1) \in \{0,1\} \times \{0,1\}$ from P_2. For every $3 \leq i \leq n$, $\mathsf{dSelPri}$ receives $(z^i_0, z^i_1) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$ from P_i. dSel delivers $(y_\alpha \oplus r, \{z^i_{y_\alpha \oplus r}\}_{3 \leq i \leq n})$ to every party (and this is where $\mathsf{dSelPri}$ differs from dSel). Let x_i be the input of party P_i to dSel (note that x_i for different parties maybe of different lengths) and let $\mathcal{S}$ be the adversary. The functionality $\mathcal{F}_{\mathsf{dSel}}$ proceeds as follows:

1. For each $i \in [3, n]$, P_i (and $\mathcal{S}$ on behalf of P_i if P_i is corrupted) sends $(\mathsf{input}, \mathsf{sid}, P_i, x_i)$ to the functionality.
2. If either of P_1 or P_2 is honest, then for each $i \in \{1, 2\}$, P_i (and $\mathcal{S}$ on behalf of P_i if P_i is corrupted) sends $(\mathsf{input}, \mathsf{sid}, P_i, x_i)$ to the functionality.
3. If P_1 and P_2 are both corrupted, $\mathcal{S}$ sends $(\mathsf{Corrupt}, \mathsf{sid}, \beta)$ where $\beta \in \{0,1\}$.
4. Upon receiving the inputs from all parties, the functionality computes:

$$\mathsf{out} := \begin{cases} (y_\alpha \oplus r, \{z^i_{y_\alpha \oplus r}\}_{i \in [3,n]}) & \text{If } P_1 \text{ or } P_2 \text{ is honest.} \\ (\beta, \{z^i_\beta\}_{i \in [3,n]}) & \text{If } P_1 \text{ and } P_2 \text{ are corrupt.} \end{cases}$$

5. The functionality delivers $(\mathsf{output}, \mathsf{sid}, \mathsf{out})$ to $\mathcal{S}$. On receiving $(\mathsf{generateOutput}, \mathsf{sid})$ from $\mathcal{S}$, the ideal functionality delivers $(\mathsf{output}, \mathsf{sid}, P_i, \mathsf{out})$ to every uncorrupted party P_i. On the other hand, if $\mathcal{S}$ sends $(\mathsf{abort}, \mathsf{sid})$, then the functionality sends $(\mathsf{output}, \mathsf{sid}, P_i, \mathsf{abort})$ to every uncorrupted party P_i. (And ignores the message if inputs from all parties in $\{P_1, \ldots, P_n\}$ have not been received.)

Figure 5: Functionality $\mathcal{F}_{\mathsf{dSel}}$

The main theorem we prove in this subsection is:

Theorem 5.11. *There exists a three-round protocol Π_{dSel} that UC-realizes the $\mathcal{F}_{\mathsf{dSel}}$ functionality. Π_{dSel} makes black-box use of a two-round malicious-secure OT with equivocal receiver security in the $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$-hybrid model.*

Building Π_{dSel}. To construct Π_{dSel}, at a high level, we apply the round-collapsing compiler of [GS18, GIS18] to a conforming protocol that implements a simple two-party functionality captured by OTplus in the $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$-hybrid model (Figure 6). OTplus gets two bits (α, r) from the receiver and two bits (s_0, s_1) from the sender and delivers $(s_\alpha \oplus r)$ to both parties. An information-theoretic protocol for securely computing the function OTplus in the $\mathcal{F}_{\kappa\text{-LeakyOT}}$-hybrid model is guaranteed from an OT combiner protocol [HKN+05, CDFR17] followed by a secure computation protocol in the OT-hybrid model [Kil88, IKO+11]. Specifically,

Theorem 5.12 ([CDFR17, Kil88, IKO+11])**.** *Let $\kappa = \Omega(\lambda)$ and consider the $\mathcal{F}_{\kappa\text{-LeakyOT}}$ functionality described in Figure 6. There exists a statistically secure protocol that UC-realizes the $\mathcal{F}_{\mathsf{OTplus}}$ functionality making a single call to the $\mathcal{F}_{\kappa\text{-LeakyOT}}$ functionality. Furthermore, the inputs to $\mathcal{F}_{\kappa\text{-LeakyOT}}$ given by an honest receiver in the above protocol are uniformly chosen $(\alpha_1, \ldots, \alpha_\kappa)$ and the inputs given by an honest sender are $(\varnothing, \{(s_0^i, s_1^i)\}_{i\in[\kappa]})$ where $\{(s_0^i, s_1^i)\}_{i\in[\kappa]}$ are uniformly chosen.*

Functionality $\mathcal{F}_{\kappa\text{-LeakyOT}}$

Let $\mathcal{S}$ be an adversary corrupting at most one among $\{P_1, P_2\}$.

- A party P_1 (and $\mathcal{S}$ on behalf of P_1 if P_1 is corrupted) sends $(\mathsf{receiver}, \mathsf{sid}, P_1, \alpha_1, \ldots, \alpha_\kappa)$.
- Another party P_2 (and $\mathcal{S}$ on behalf of P_2 if P_2 is corrupted) sends $(\mathsf{sender}, \mathsf{sid}, P_2, (K, \{(s_0^i, s_1^i)\}_{i\in[\kappa]}))$ to the functionality where $K \subseteq [\kappa]$ is a set of size at most λ and $s_b^i \in \{0,1\}$ for each $i \in [\kappa]$ and $b \in \{0,1\}$.
- On receiving both these messages, the functionality computes $\mathsf{out}_1 := \{(\alpha_i, s_{\alpha_i}^i)\}_{i\in[\kappa]}$ and $\mathsf{out}_2 := \{\alpha_i\}_{i\in K}$.
- For $i \in \{1,2\}$, if P_i is corrupted, the functionality delivers $(\mathsf{output}, \mathsf{sid}, P_i, \mathsf{out}_i)$ to $\mathcal{S}$. On receiving $(\mathsf{generateOutput}, \mathsf{sid})$ from $\mathcal{S}$ (if either of P_1 or P_2 is corrupted), the functionality delivers $(\mathsf{output}, \mathsf{sid}, P_i, \mathsf{out}_i)$ to every honest P_i. On the other hand, if $\mathcal{S}$ sends $(\mathsf{abort}, \mathsf{sid})$, it sends $(\mathsf{output}, \mathsf{sid}, P_i, \mathsf{abort})$ to every honest P_i.

Figure 6: Functionality $\mathcal{F}_{\kappa\text{-LeakyOT}}$

The above theorem implies that a protocol for realizing $\mathcal{F}_{\mathsf{OTplus}}$ has the following structure:

- **Call to $\mathcal{F}_{\kappa\text{-LeakyOT}}$ functionality.** The honest P_1 samples uniform bits $(\alpha_1, \ldots, \alpha_\kappa)$ as input to the functionality. The honest P_2 samples uniform bits $\{(s_0^i, s_1^i)\}_{i\in[\kappa]}$ and sends $(\varnothing, \{(s_0^i, s_1^i)\}_{i\in[\kappa]})$ to the functionality.
- **Protocol Π_{OTplus}.** Using the output of $\mathcal{F}_{\kappa\text{-LeakyOT}}$ functionality, P_1 and P_2 interact with each other using the *statistically-secure* protocol Π_{OTplus} (from Theorem 5.12) that realizes the $\mathcal{F}_{\mathsf{OTplus}}$ functionality. In this protocol, P_1's input is given by $((\alpha, r), (s_{\alpha_1}^1, \alpha_1), \ldots, (s_{\alpha_\kappa}^\kappa, \alpha_\kappa))$ and P_2's input is given by $((y_0, y_1), (s_0^1, s_1^1), \ldots, (s_0^\kappa, s_1^\kappa))$ (where (α, r) are the P_1's inputs to the $\mathcal{F}_{\mathsf{OTplus}}$ functionality and y_0, y_1 are P_2's inputs). Without loss of generality, we assume that the last message from P_1 to P_2 contains the output of $\mathcal{F}_{\mathsf{OTplus}}$.

Let Φ be the conforming protocol obtained as result of the transformation given in Theorem 5.10 to the protocol Π_{OTplus}. We assume without loss of generality that the input of P_1 in Φ is of the form $(s_{\alpha_1}^i, \ldots, s_{\alpha_\kappa}^i, \alpha_1, \ldots, \alpha_k, \alpha, r)$ and that of P_2 is $(\{s_0^i, s_1^i\}_{i\in[\kappa]}, y_0, y_1)$. We further assume w.l.o.g. that at the end of the computation phase of Φ, $\mathsf{st}[\ell/2]$ (for each $i \in \{1,2\}$) contains the output of the protocol (i.e., $v_1[\ell/2] = v_2[\ell/2] = 0$) and post just outputs this bit. We now present an informal description of Π_{dSel}.

Informal Description of Π_{dSel}. As mentioned earlier, Π_{dSel} is obtained by applying the round-collapsing compiler of [GS18, GIS18] to the conforming protocol Φ using $\Pi^{\dagger}_{\mathsf{dSelPri}}$ to implement the double-selection functionality. However, the main challenge is that $\Pi^{\dagger}_{\mathsf{dSelPri}}$ suffers from an input-dependent abort issue and we need a mechanism to overcome this. Towards this goal, in Π_{dSel}, we run κ copies of $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ with the input of P_1 in the k-th copy being $\{\alpha_k, v_1[k]\}_{k\in[\kappa]}$ (where v_1 is the private state of P_1 as per the round-collapsing compiler and α_k is uniformly chosen), the input of P_2 being a random pair of bits (s^k_0, s^k_1) and the inputs for the rest of parties being equal to a pair of secret keys for a SKE scheme (the role of these keys will be clear soon). These κ-executions of $\Pi^{\dagger}_{\mathsf{dSelPri}}$ lead to P_1 and P_2 sharing κ-random OT correlations. It is these κ-random OT correlations that serve as the input and output of the leaky OT functionality. Specifically, as argued in the proof, we show that a corrupt P_2 cannot guess more than λ among $(\alpha_1, \ldots, \alpha_\kappa)$ without triggering an abort by an honest P_1 with overwhelming probability. In other words, the size of the set K that a corrupt P_2 sends to the $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$ functionality is at most λ. This allows us to use the security of the conforming protocol Φ to argue the security of the round-collapsed protocol.

Having defined the inputs to Φ, we now discuss how the labels corresponding to the initial public joint state for every party's garbled circuit are made available in 3 rounds. Note that the part of the public state that corresponds to the OT correlation given to P_1 is revealed to P_1 only by the end of round-2 by the $\Pi^{\dagger}_{\mathsf{dSelPri}}$-functionality. Thus, P_1 can send its labels corresponding to the joint public state in round-3. However, this poses a challenge for the other parties as they do not learn this value by the beginning of round-3. This is where we use the secret keys used in the calls to $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$. Recall that P_1 gets P_j's secret key corresponding to the bit $s^k_{\alpha_k} \oplus v_1[k]$ from $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ functionality at the end of round-2. In round-3, P_1 sends this secret key and P_j sends a pair of encryptions, encrypting b-th label under b-th key for $b \in \{0, 1\}$. Putting these two things together, all parties can recover the label for P_j's circuit corresponding to the bit $s^k_{\alpha_k} \oplus v_1[k]$. This way all the parties obtain the labels for the inital joint public state for the first set of garbled circuits. This will trigger evaluation of the bunch of circuits emulating Φ.

The garbled circuits generated by P_1 and P_2 will perform the interaction as dictated by the protocol Φ while the garbled circuits generated by all other parties will listen to this interaction. By the virtue of listening to this interaction, the last garbled circuit of every party in $\{P_3, \ldots, P_n\}$ will output the labels for st that has $(s_\alpha \oplus r)$ at the position $\ell/2$. We now introduce another layer of garbled circuits for only P_3 to P_n that take the labels for st, hard-wires z^i_0, z^i_1 and outputs $z^i_{\mathsf{st}[\ell/2]}$ if st does not indicate an abort of P_1 or P_2. Without loss of generality, we can assume that st contains this information on abort. [5]

Lastly, in the formal description, we consider the $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ functionality instantiated with $n+1$ parties with party P_2 additionally playing the role of P_{n+1}. Specifically, the inputs of party P_2 includes (y_0, y_1) as well as (z^2_0, z^2_1). We give the description of the first three rounds of the protocol Π_{dSel} in the $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$-hybrid model in Figure 7.

Protocol Π_{dSel}

Inputs: P_1 inputs $(\alpha, r) \in \{0,1\} \times \{0,1\}$, P_2 inputs $(y_0, y_1) \in \{0,1\} \times \{0,1\}$. For every $3 \leq i \leq n$, P_i

[5] To tackle a malicious behaviour of P_i, we make them commit to z^i_0, z^i_1 via OT receiver messages in the first round and reveal the opening information via the garbled circuit.

inputs $(z_0^i, z_1^i) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$.

Output: Every party outputs $(y_\alpha \oplus r, \{z^i_{y_\alpha \oplus r}\}_{3\leq i\leq n})$ and the other parties do not get any outputs.

Primitives and Functionalities: (a) A malicious-secure, two-round OT with equivocal receiver security defined by $(K_{\mathsf{OT}}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ (see Section 3.3). We use OT_1^* to denote an algorithm that takes a crs and $q(\lambda)$-bit string (for some polynomial $q(\cdot)$) as input and applies OT_1 to each bit of that string. (b) Functionality $\mathcal{F}^\dagger_{\mathsf{dSelPri}}$. (c) The conforming protocol Φ obtained as a result of the transformation in Theorem 5.10 to Π_{OTplus} as discussed. (d) Garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$ (see Section 3.2) (e) A symmetric-key Encryption Scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

Common Random/Reference String: For each $i \in [n]$, sample $\mathsf{crs}^i \leftarrow K_{\mathsf{OT}}(1^\lambda)$ and output $\{\mathsf{crs}^i\}_{i\in[n]}$ as the common random/reference string.

Round-1: In the first round,

- Parties P_1 and P_2 run $\mathsf{pre}(1^\lambda, 1)$ and $\mathsf{pre}(1^\lambda, 2)$ to get v_1 and v_2 respectively. For each $i \in [3, n]$, P_i sets $v_i = 0^\ell$.
- P_1 chooses κ random bits $\alpha_1, \ldots, \alpha_\kappa$ and P_2 chooses random pairs of bits (s_0^k, s_1^k) for each $k \in [\kappa]$.
- For each $i \in [2, n]$ and for each $k \in [\kappa]$, P_i chooses two random secret keys $(sk_0^{i,k}, sk_1^{i,k})$ using $\mathsf{Gen}(1^\lambda)$.
- For each $k \in [\kappa]$, P_1 sends $(\mathsf{input}, k, P_1, (\alpha_k, v_1[k]))$, P_2 sends $(\mathsf{input}, k, P_2, (s_0^k, s_1^k))$ and for each $i \in [2, n]$, P_i sends $(\mathsf{input}, k, P_i, (sk_0^{i,k}, sk_1^{i,k}))$ to $\mathcal{F}^\dagger_{\mathsf{dSelPri}}$.
- For each $i \in [3, n]$, for each $b \in \{0, 1\}$, P_i computes $(\mathsf{otr}_b^i, \mu_b^i) \leftarrow \mathsf{OT}_1^*(\mathsf{crs}^i, z_b^i)$.
- For each $i \in [3, n]$, P_i broadcasts $\{\mathsf{otr}_b^i\}_{b\in\{0,1\}}$ to every other party.

Round-2: In the second round,

- P_1 sets $x_1^{\mathsf{part}} := (\alpha_1, \ldots, \alpha_\kappa, \alpha, r)$ and P_2 sets $x_2 := (\{s_0^k, s_1^k\}_{k\in[\kappa]}, y_0, y_1)$.
- P_1 and P_2 respectively set $z_1^{\mathsf{part}} := (x_1^{\mathsf{part}} \oplus v_1[\kappa+1, 2\kappa+2])\|0^{\ell/2-(2\kappa+2)}$ and $z_2 := (x_2 \oplus v_2[\ell/2 + 1, \ell/2 + 2\kappa + 2])\|0^{\ell/2-(2\kappa+2)}$.
- For each $i \in \{1, 2\}$ and for each t such that $\phi_t = (i, f, g, h)$ (A_i is the set of such values of t), for each $\alpha, \beta \in \{0, 1\}$, P_i computes: $(\mathsf{otr}^{i,t,\alpha,\beta}, \mu^{i,t,\alpha,\beta}) \leftarrow \mathsf{OT}_1(\mathsf{crs}^i, v_i[h] \oplus \mathsf{NAND}(v_i[f] \oplus \alpha, v_i[g] \oplus \beta))$.
- P_1 broadcasts $\left(z_1^{\mathsf{part}}, \{\mathsf{otr}^{i,t,\alpha,\beta}\}_{t\in A_1,\alpha,\beta\in\{0,1\}}\right)$ and P_2 broadcasts $\left(z_2, \{\mathsf{otr}^{i,t,\alpha,\beta}\}_{t\in A_2,\alpha,\beta\in\{0,1\}}\right)$ to every other party.

Round-3: In the final round, each party P_i does the following:

- If $i = 1$, P_1 receives for each $k \in [\kappa]$, $(\mathsf{output}, k, P_1, (x_1[k], \{sk^{i,k}_{x_1[k]\oplus v_1[k]}\}_{i\in[2,n]}))$ from $\mathcal{F}^\dagger_{\mathsf{dSelPri}}$ where $x_1[k] = s^k_{\alpha_k}$.[a]
- P_i sets $\mathsf{st} := 0^\kappa\|\left(z_1^{\mathsf{part}}\|z_2\right)$.
- If $i \in [3, n]$, P_i computes $(\widetilde{\mathsf{ChkC}}^i, \mathsf{lab}^{i,T+1}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{ChkC}^i[\{z_b^i, \mu_b^i\}_{b\in\{0,1\}}])$.
- If $i \in \{1, 2\}$, P_i sets $\mathsf{lab}^{i,T+1} = \{\perp, \perp\}_{k\in[\ell]}$.
- **for** each t from T down to 1,
  1. Parse ϕ_t as (i^*, f, g, h).
  2. If $i = i^*$ then it computes (where $C^{i,t}$ is described in Figure 9) $\left(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}\right) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \{\mu^{i,t,\alpha,\beta}\}_{\alpha,\beta}, \perp, \mathsf{lab}^{i,t+1}])$.

3. If $i \neq i^*$ then for every $\alpha, \beta \in \{0,1\}$, it sets $\mathsf{ots}^{i^*,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{crs}^{i^*}, \mathsf{otr}^{i^*,t,\alpha,\beta}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$ and computes $\left(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}\right) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \perp, \{\mathsf{ots}^{i^*,t,\alpha,\beta}\}_{\alpha,\beta}, \mathsf{lab}^{i,t+1}])$.

- Each P_i sends $\left(\{\widetilde{C}^{i,t}\}_{t\in[T]}, \{\mathsf{lab}^{i,1}_{k,\mathsf{st}[k]}\}_{k\in[\kappa+1,\ell]}\right)$ to every other party and if $i \in [3,n]$, it also sends $\widetilde{\mathsf{ChkC}}^i$. In addition, P_1 sends $\left\{\mathsf{lab}^{1,1}_{k,x_1[k]\oplus v_1[k]}, x_1[k]\oplus v_1[k], \{sk^{i,k}_{x_1[k]\oplus v_1[k]}\}_{i\in[2,n]}\right\}_{k\in[\kappa]}$ and for each $i \in [2,n]$, P_i sends $\{\mathsf{Enc}(sk^{i,k}_0, \mathsf{lab}^{i,1}_{k,0}), \mathsf{Enc}(sk^{i,k}_1, \mathsf{lab}^{i,1}_{k,1})\}_{k\in[\kappa]}$.

Output. Each party P_i does the following:

- It sets $\mathsf{st}[k] = x_1[k] \oplus v_1[k]$ for each $k \in [\kappa]$ receiving the value from P_1's broadcast.
- For each $j \in [2,n]$ and $k \in [\kappa]$, it recovers $\mathsf{lab}^{j,1}_{k,\mathsf{st}[k]} \leftarrow \mathsf{Dec}(sk^{j,k}_{\mathsf{st}[k]}, \mathsf{Enc}(sk^{i,k}_{\mathsf{st}[k]}, \mathsf{lab}^{i,1}_{k,\mathsf{st}[k]}))$.
- Let $\widetilde{\mathsf{lab}}^{1,1} := \left\{\{\mathsf{lab}^{1,1}_{k,x_1[k]\oplus v_1[k]}\}_{k\in[\kappa]}, \{\mathsf{lab}^{1,1}_{k,\mathsf{st}[k]}\}_{k\in[\kappa+1,\ell]}\right\}$.
- For each $j \in [2,n]$, let $\widetilde{\mathsf{lab}}^{j,1} := \{\mathsf{lab}^{j,1}_{k,\mathsf{st}[k]}\}_{k\in[\ell]}$.
- **for** each t from 1 to T do:
  1. Parse ϕ_t as (i^*, f, g, h).
  2. Compute $((\alpha, \beta, \gamma), \mu, \widetilde{\mathsf{lab}}^{i^*,t+1}) := \mathsf{Eval}(\widetilde{C}^{i^*,t}, \widetilde{\mathsf{lab}}^{i^*,t})$.
  3. Set $\mathsf{st}[h] := \gamma$.
  4. **for** each $j \neq i^*$ do:
     (a) Compute $(\mathsf{ots}, \{\mathsf{lab}^{j,t+1}_k\}_{k\in[\ell]\setminus\{h\}}) := \mathsf{Eval}(\widetilde{C}^{j,t}, \widetilde{\mathsf{lab}}^{j,t})$.
     (b) Recover $\mathsf{lab}^{j,t+1}_h := \mathsf{OT}_3(\mathsf{crs}^{i^*}, \mathsf{ots}, (\gamma, \mu))$.
     (c) Set $\widetilde{\mathsf{lab}}^{j,t+1} := \{\mathsf{lab}^{j,t+1}_k\}_{k\in[\ell]}$.
- For each $j \in [3,n]$,
  - Compute $(z^j, \mu^j) := \mathsf{Eval}(\widetilde{\mathsf{ChkC}}^j, \widetilde{\mathsf{lab}}^{j,T+1})$
  - Run $\mathsf{CheckValid}(\mathsf{crs}^j, \mathsf{otr}^j_{\mathsf{st}[\ell/2]}, (z^j, \mu^j))$.
- If any of runs of the CheckValid algorithm outputs 0 then abort. Otherwise, output $(\mathsf{st}[\ell/2], \{z^j_{\mathsf{st}[\ell/2]}\}_{j\in[3,n]})$.

[a] Note that this message is received in the end of round-2, since $\Pi^\dagger_{\mathsf{dSelPri}}$ is a 2-round protocol.

Figure 7: Protocol Π_{dSel}

Circuit $C^{i,t}$

Input. st

Hard-coded Information. v_i, $\{\mu^{i,t,\alpha,\beta}\}_{\alpha,\beta}$, $\{\mathsf{ots}^{t,\alpha,\beta}\}_{\alpha,\beta}$ and $\mathsf{lab} = \{\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}\}_{k\in[\ell]}$.

- Let $\phi_t = (i^*, f, g, h)$.
- **if** $i = i^*$ **then**:
  - Compute $\mathsf{st}[h] := \mathsf{NAND}(\mathsf{st}[f] \oplus v_i[f], \mathsf{st}[g] \oplus v_i[g]) \oplus v_i[h]$.

– Output $((\mathsf{st}[f], \mathsf{st}[g], \mathsf{st}[h]), \mu^{i,t,\mathsf{st}[f],\mathsf{st}[g]}, \{\mathsf{lab}_{k,\mathsf{st}[k]}\}_{k\in[\ell]})$.

- **else**:

  – Output $(\mathsf{ots}^{i^*,t,\mathsf{st}[f],\mathsf{st}[g]}, \{\mathsf{lab}_{k,\mathsf{st}[k]}\}_{k\in[\ell]\setminus\{h\}})$.

Figure 8: Circuit $C^{i,t}$

Circuit ChkC^i

Input. st

Hard-coded Information. $\{z_b^i, \mu_b^i\}_{b\in\{0,1\}}$.

- Check from st if P_1 or P_2 have not aborted. We assume w.l.o.g. that this information is public from st.
- If no abort occurs, then output $z^i_{\mathsf{st}[\ell/2]}, \mu^i_{\mathsf{st}[\ell/2]}$. Otherwise, output $\perp$.

Figure 9: Circuit ChkC^i

Lemma 5.13. *Let $\mathcal{A}$ be an (possibly malicious) adversary corrupting an arbitrary subset of parties in the protocol Π_{dSel}. There exists a simulator Sim such that for any environment $\mathcal{Z}$,*

$$\mathrm{EXEC}_{\mathcal{F}_{\mathsf{dSel}},\mathsf{Sim},\mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}_{\Pi_{\mathsf{dSel}},\mathcal{A},\mathcal{Z}}$$

Proof. Let $C \subset \{P_1, \ldots, P_n\}$ be the set of parties corrupted by $\mathcal{A}$ and let $H = \{P_1, \ldots, P_n\} \setminus C$ denote the set of honest parties. Since we assume that $\mathcal{A}$ is static, the set of corrupted parties C is decided before the beginning of the protocol. We now give the description of the ideal world simulator Sim. Sim internally uses the simulators $(\mathsf{Sim}_R, \mathsf{Sim}_S)$ of the oblivious transfer (see Section 3.3), Sim_Φ of the conforming protocol Φ, and the simulator for garbled circuit $\mathsf{Sim}_{\mathsf{GC}}$.

Interaction with environment $\mathcal{Z}$. For every input value corresponding to the set of corrupted parties C that Sim receives from the environment $\mathcal{Z}$, Sim writes this value to $\mathcal{A}$'s input tape. Similarly, the contents of $\mathcal{A}$'s output tape is written to Sim's output tape. We now describe how Sim simulates the interaction of honest parties with $\mathcal{A}$.

Simulating the interaction with $\mathcal{A}$: For every concurrent interaction with the session identifier sid that $\mathcal{A}$ may start, the simulator does the following:

Common Random/Reference String Generation: Sim generates the crs as follows:

- For each $i \in [n]$, if $P_i \in C$, then Sim samples $\underline{(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}_S^1(1^\lambda)}$. Else, it samples $\underline{(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}_R^1(1^\lambda)}$.
- Sim sets the crs to be $(\mathsf{crs}^1, \ldots, \mathsf{crs}^n)$.

Round-1 message from Sim to $\mathcal{A}$. To generate the round-1 message, Sim does the following:

- For each $i \in [3, n]$, if $P_i \in H$, for each $b \in \{0,1\}$, Sim computes $(\mathsf{otr}^i_b, \{\mu^{i,k}_{b,0}, \mu^{i,k}_{b,1}\}_{k\in[\lambda]}) \leftarrow \mathsf{Sim}^2_R(\mathsf{crs}^i, \mathsf{td}^i)$.[6]
- Sim sends the first round messages on behalf of the honest parties to $\mathcal{A}$.

Round-2 message from Sim to $\mathcal{A}$. For each $i \in \{1, 2\}$ such that $P_i \in H$, Sim does the following:

- It sets $z_i := r_i \| 0^{\ell/2-(2\kappa+2)}$ where r_i is chosen uniformly from $\{0,1\}^{2\kappa+2}$.
- If $i = 1$, it sets $z_1^{\mathsf{part}} = z_1[\kappa + 1, \ell/2]$.
- For each $t \in A_i$ and for each $\alpha, \beta \in \{0, 1\}$, it generates $(\mathsf{otr}^{i,t,\alpha,\beta}, \mu_0^{i,t,\alpha,\beta}, \mu_1^{i,t,\alpha,\beta}) \leftarrow \mathsf{Sim}^2_R(\mathsf{crs}^i, \mathsf{td}^i)$.
- Sim sends the second round message on behalf of the honest parties to $\mathcal{A}$.

Extraction and Faithful execution. Sim does the following:

- **Computing z_1 if $P_1 \in C$:**
  - Sim intercepts the message $(\mathsf{input}, k, P_1, (\alpha_k, v_1[k]))$ that $\mathcal{A}$ sends to $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ for each $k \in [\kappa]$.
  - If $P_2 \in C$, Sim intercepts the message $(\mathsf{input}, k, P_2, (s^k_0, s^k_1))$ that $\mathcal{A}$ sends to $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ for each $k \in [\kappa]$.
  - If $P_2 \in H$, Sim chooses a uniform random bit $s^k_{\alpha_k}$ for each $k \in [\kappa]$.
  - Sim sets $z_1 := (s^1_{\alpha_1} \oplus v_1[1], \ldots, s^\kappa_{\alpha_\kappa} \oplus v_1[\kappa]) \| z_1^{\mathsf{part}}$.
- **Extraction.**
  - For each $i \in \{1, 2\}$ such that $P_i \in C$ and for each $t \in A_i$, $\alpha, \beta \in \{0, 1\}$, Sim computes $b^{i,t,\alpha,\beta} = \mathsf{Sim}^2_S(\mathsf{crs}^i, \mathsf{td}^i, \mathsf{otr}^{i,t,\alpha,\beta})$.
  - Additionally, for each $i \in [3, n]$ such that $P_i \in C$ and for each $b \in \{0, 1\}$, Sim computes $z^i_b := \mathsf{Sim}^2_S(\mathsf{crs}^i, \mathsf{td}^i, \mathsf{otr}^i_b)$. It sends $(\mathsf{input}, \mathsf{sid}, P_i, (z^i_0, z^i_1))$ to the ideal functionality on behalf of corrupt P_i.
- **Faithful Interaction.** We define an interactive procedure $\mathsf{Faithful}(i, \{z_i\}_{i\in\{1,2\}}, \{b^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta})$ that on input $i \in \{1, 2\}$, $\{z_i\}_{i\in\{1,2\}}$, $\{b^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}$ (z_1 as extracted above and z_2 as received from P_2 in a round-2 broadcast) produces protocol Φ message on behalf of party P_i (acting consistently/faithfully with the extracted values) as follows:
  - Set $\mathsf{st}^* := z_1 \| \ldots \| z_n$.
  - For $t \in \{1 \cdots T\}$

[6] Here, we slightly abuse the notation and use Sim^2_R to compute the output of OT^*_1.

    * Parse $\phi_t = (i^*, f, g, h)$.
    * If $i \neq i^*$ then it waits for a bit from P_{i^*} and sets $\mathsf{st}^*[h]$ to be the received bit once it is received.
    * Set $\mathsf{st}^*[h] := b^{i^*,t,\mathsf{st}^*[f],\mathsf{st}^*[g]}$ and output it to all the other parties.

- **If P_1 and P_2 are in C.**
  - Sim obtains the transcript Z of Φ by implementing the messages sent by corrupt P_i for $i \in \{1,2\}$ using $\mathsf{Faithful}(i, \{z_i\}_{i\in[2]}, \{b^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta})$.
  - Let st^*_T be the state at the end of the faithful execution of one of the corrupt parties (this value is the same for all corrupt parties). It sends $(\mathsf{Corrupt}, \mathsf{sid}, \delta := \mathsf{st}^*_T[\ell/2])$ to the ideal functionality.
  - Sim receives $(\delta, \{z^i_\delta\}_{i\in[3,n]})$ from the ideal functionality.
  - **OT Receiver Equivocation for the inputs of honest P_i for $i \in [3,n]$.** For each $i \in [3,n]$ such that $P_i \in H$, Sim sets $\mu^i := \{\mu^{i,k}_{\delta,z^i_\delta[k]}\}_{k\in[\lambda]}$. This step ensures that corrupt P_1 finds the correct $\{z^i_\delta\}_{i\in[3,n]}$ via the last layer of garbled circuits.

Round-3 message from Sim to $\mathcal{A}$. To generate the round-3 message Sim does the following:

- Initialize $\mathsf{aux} = \perp$. Here, aux is used to denote the inputs and outputs of $\mathcal{A}$ implicitly gives to the $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$-functionality when interacting with an honest party.
- **Updating the value of aux.**
  - **If $P_1 \in H$ and $P_2 \in C$.**
    * For each $k \in [\kappa]$, it intercepts the message $(\mathsf{input}, k, P_2, (s^i_0, s^i_1))$ that $\mathcal{A}$ sends on behalf of corrupt P_2 to $\mathcal{F}^\dagger_{\mathsf{dSelPri}}$.
    * For each $k \in [\kappa]$, Sim additionally intercepts the message $(\mathsf{predicate}, k, \mathsf{EQ}_{\beta_k})$ that $\mathcal{A}$ might send to the $\mathcal{F}^\dagger_{\mathsf{dSelPri}}$ functionality. If the number of such k for which $\mathcal{A}$ sends this message is greater than λ, then Sim sends $(\mathsf{abort}, \mathsf{sid})$ to the functionality $\mathcal{F}_{\mathsf{dSel}}$.
    * On the other hand, if the number of such k for which $\mathcal{A}$ sends this message is less than or equal to λ, Sim does the following:
      · Let K be the subset of $[\kappa]$ such that $\mathcal{A}$ sends the message $(\mathsf{predicate}, k, \mathsf{EQ}_{\beta_k})$.
      · For every $k \in K$, it chooses a uniform bit α_k.
      · If for any such k, $\alpha_k = \beta_k$ then, Sim sends $(\mathsf{abort}, \mathsf{sid})$ to the functionality $\mathcal{F}_{\mathsf{dSel}}$.
      · Else, it sets $\mathsf{aux} := (K, \{\alpha_k\}_{k\in K}, \{(s^i_0, s^i_1)\}_{i\in[\kappa]})$.
  - **If $P_1 \in C$ and $P_2 \in H$.**
    * Sim sets $\mathsf{aux} := \{(\alpha_k, s^k_{\alpha_k})\}_{k\in[\kappa]}$ where $s^k_{\alpha_k}$ was the bit that was randomly chosen while computing z_1.
- **If P_1 or P_2 is in H.**
  - For each $i \in \{1,2\}$ such that $P_i \in C$, Sim sends z_i to Sim_Φ on behalf of the corrupted party P_i. It also initializes Sim_Φ with the value aux. This starts the computation phase of Φ with the simulator Sim_Φ.

  - Sim provides computation phase messages from corrupted parties to Sim_Φ by following a faithful execution. More formally, for every corrupted party P_i where $i \in \{1,2\}$, Sim generates messages on behalf of P_i for Sim_Φ using the procedure $\mathsf{Faithful}(i, \{z_i\}_{i\in[2]}, \{b^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta})$.
  - At some point during the execution, Sim_Φ will return the extracted inputs $\{x_i\}_{i\in C\cap\{P_1,P_2\}}$ of the corrupted parties. For each $i \in C \cap \{P_1, P_2\}$, Sim sends $(\mathsf{input}, \mathsf{sid}, P_i, x_i)$ to the ideal functionality and obtains the output $(\delta := y_\alpha \oplus r, \{z^i_\delta\}_{i\in[3,n]})$. It sends δ as the output to Sim_Φ. Sim_Φ completes the rest of the execution of the protocol.
  - Let $\mathsf{Z} \in \{0,1\}^t$ where Z_t is the bit sent in the t^{th} round of the computation phase of Φ be output of this execution. And let st^*_T be the state value at the end of faithful execution of one of the corrupted parties (this value is the same for all the parties). Also, set for each $t \in \cup_{i\in H\cap\{P_1,P_2\}} A_i$ and $\alpha, \beta \in \{0,1\}$ set $\mu^{i,t,\alpha,\beta} := \mu^{i,t,\alpha,\beta}_{\mathsf{Z}_t}$. The last step here corresponds to OT receiver equivocation so that the bit opened in tth round is as per the simulation of Φ with Sim_Φ.
  - For each $i \in [3,n]$ such that $P_i \in H$, Sim sets $\mu^i := \{\mu^{i,k}_{\delta,z^i_\delta[k]}\}_{k\in[\lambda]}$.

- For each $i \in [2,n]$, if $P_i \in H$, Sim chooses $sk^{i,k}$ uniformly from $\mathsf{Gen}(1^\lambda)$. For each $i \in [2,n]$, if $P_i \in C$, then Sim intercepts the message $(\mathsf{input}, k, P_i, (sk^{i,k}_0, sk^{i,k}_1))$ that $\mathcal{A}$ sends to $\mathcal{F}^\dagger_{\mathsf{dSelPri}}$ for each $k \in [\kappa]$ and sets $sk^{i,k} := sk^{i,k}_{z_1[k]}$. If $P_1 \in C$, it delivers $(\mathsf{output}, k, (z_1[k], \{sk^{i,k}\}_{i\in[2,n]})$ for each $k \in [\kappa]$.

- For each $i \in [n]$ such that $P_i \in H$, Sim does the following:
  - If $i \in [3,n]$, it computes $\left(\widetilde{\mathsf{ChkC}}^i, \{\mathsf{lab}^{i,T+1}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, 1^{|\mathsf{ChkC}^i|}, 1^\ell, \mathsf{out}_i)$ where out_i is $\perp$ if st^*_T leads to an abort of either P_1 or P_2 and is otherwise, equals to (z^i_δ, μ^i).
  - If $i \in \{1,2\}$, it sets $\mathsf{lab}^{i,T+1}_k = \perp$ for every $k \in [\ell]$.
  - **for** each t from T down to 1,
    * Parse ϕ_t as (i^*, f, g, h).
    * Set $\alpha^* := \mathsf{st}^*_T[f]$, $\beta^* := \mathsf{st}^*_T[g]$, and $\gamma^*_T := \mathsf{st}^*[h]$.
    * If $i = i^*$ then computes
    $$\left(\widetilde{C}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left((\alpha^*, \beta^*, \gamma^*), \mu^{i,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_k\}_{k\in[\ell]}\right)\right).$$
    * If $i \neq i^*$ then set $\mathsf{ots}^{i^*,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2(\mathsf{crs}^{i^*}, \mathsf{otr}^{i^*,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_h, \mathsf{lab}^{i,t+1}_h)$ and computes
    $$\left(\widetilde{C}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left(\mathsf{ots}^{i^*,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_k\}_{k\in[\ell]\setminus\{h\}}\right)\right).$$
  - It sends $\left(\{\widetilde{C}^{i,t}\}_{t\in[T]}, \{\mathsf{lab}^{i,1}_k\}_{k\in[\kappa+1,\ell]}\right)$ to $\mathcal{A}$. If $i \in [3,n]$, then Sim sends $\widetilde{\mathsf{ChkC}}^i$.
  - If $i = 1$, Sim additionally sends $(\{\mathsf{lab}^{1,1}_k, z_1[k], sk^{i,k}\}_{k\in[\kappa]})$. Else, Sim chooses a uniform key $\overline{sk}^{i,k}$ using $\mathsf{Gen}(1^\lambda)$ and sets $sk^{i,k}_{z_1[k]} = sk^{i,k}$ and $sk^{i,k}_{1-z_1[k]} = \overline{sk}^{i,k}$. Sim sends $\{\mathsf{Enc}(sk^{i,k}_0, \mathsf{lab}^{i,1}_k), \mathsf{Enc}(sk^{i,k}_1, \mathsf{lab}^{i,1}_k)\}_{k\in[\kappa]}$.

Output Computation. For every $i \in [n] \setminus H$, Sim obtains the second round message from $\mathcal{A}$ on behalf of the malicious parties. Subsequent to obtaining these messages, Sim uses the honest output computing procedure to see if the execution of garbled circuits proceeds consistently with the expected faithful execution. If the computation succeeds then, Sim sends $(\mathsf{generateOutput}, \mathsf{sid})$ to the ideal functionality. Otherwise, it sends $(\mathsf{abort}, \mathsf{sid})$.

Proof of Indistinguishability. We now show that the real execution and the simulated execution are computationally indistinguishable via a hybrid argument.

- $\underline{\mathsf{Hybrid}_0}$: This corresponds to the view of the adversary and the output of the honest parties in the real execution of the protocol.
- $\underline{\mathsf{Hybrid}_1}$: In this hybrid, we make the following changes:
  - **CRS Generation.** For each $i \in [3, n]$, if $P_i \in H$, we sample $(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}^1_R(1^\lambda)$ and if $P_i \in C$, we sample $(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}^1_S(1^\lambda)$. We use the above sampled crs^i to generate the crs.
  - **Round-1 message from honest P_i where $i \in [3, n]$.** For every $i \in [3, n]$ such that P_i is honest and for each $b \in \{0, 1\}$, we compute $(\mathsf{otr}^i_b, \{\mu^{i,k}_{b,0}, \mu^{i,k}_{b,1}\}_{k \in [\lambda]}) \leftarrow \mathsf{Sim}^2_R(\mathsf{crs}^i, \mathsf{td}^i)$.
  - **Input Extraction.** For each $i \in [3, n]$ such that $P_i \in C$ and for each $b \in \{0, 1\}$, we compute $z^i_b := \mathsf{Sim}^2_S(\mathsf{crs}^i, \mathsf{td}^i, \mathsf{otr}^i_b)$. For every $i \in [3, n]$ such that $P_i \in C$, we send $(\mathsf{input}, \mathsf{sid}, P_i, (z^i_0, z^i_1))$ to the ideal functionality.
- $\underline{\mathsf{Hybrid}_2}$: In this hybrid, we make the following changes:
  - **CRS Generation.** For each $i \in \{1, 2\}$ such that $P_i \in C$, we sample $(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}^1_S(1^\lambda)$. We use the sampled crs^i to generate the crs.
  - **Input Extraction.** For each $i \in \{1, 2\}$ such that $P_i \in C$ and for each $t \in A_i$, $\alpha, \beta \in \{0, 1\}$, we compute $b^{i,t,\alpha\beta} = \mathsf{Sim}^2_S(\mathsf{crs}^i, \mathsf{td}^i, \mathsf{otr}^{i,t,\alpha,\beta})$.
  - **Round-3 message from honest P_i where $i \in [n]$.** For each $t \in [T]$,
    * Let $\phi_t = (i^*, f, g, h)$.
    * For each $i \notin [n] \setminus \{i^*\}$ such that $P_i \in H$, compute for each $\alpha, \beta \in \{0, 1\}$, $\mathsf{ots}^{i,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{crs}^{i^*}, \mathsf{otr}^{i^*,t,\alpha,\beta}, \mathsf{lab}^{i,t}_{b^{i^*,t,\alpha,\beta}}, \mathsf{lab}^{i,t}_{\bar{b}^{i^*,t,\alpha,\beta}})$. Here, if $P_{i^*} \in C$, then $b^{i^*,t,\alpha,\beta}$ is the extracted value. Otherwise, if $P_{i^*} \in H$, then $b^{i^*,t,\alpha,\beta} = v_{i^*}[h] \oplus \mathsf{NAND}(v_{i^*}[f] \oplus \alpha, v_{i^*}[g] \oplus \beta)$.
- $\underline{\mathsf{Hybrid}_3}$: In this hybrid, we make the following changes:
  - **CRS Generation.** For each $i \in \{1, 2\}$, if $P_i \in H$, sample $(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}^1_R(1^\lambda)$ and use crs^i to generate the crs.
  - **Round-2 message:** For each $i \in \{1, 2\}$, if $P_i \in H$, and for each $t \in A_i$ and $\alpha, \beta \in \{0, 1\}$, generate $(\mathsf{otr}^{i,t,\alpha,\beta}, \mu^{i,t,\alpha,\beta}_0, \mu^{i,t,\alpha,\beta}_1) \leftarrow \mathsf{Sim}^2_R(\mathsf{crs}^i, \mathsf{td}^i)$.
- $\underline{\mathsf{Hybrid}_4}$: In this hybrid, we make the following changes:
  - **Computing z_1 when $P_1 \in C$.** Skip the following changes if $P_1 \in H$.

    * We intercept the message $(\mathsf{input}, k, P_1, (\alpha_k, v_1[k]))$ that $\mathcal{A}$ sends to $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ for each $k \in [\kappa]$.
    * If $P_2 \in C$, we intercept the message $(\mathsf{input}, k, P_2, (s^k_0, s^k_1))$ that $\mathcal{A}$ sends to $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ for each $k \in [\kappa]$.
    * If $P_2 \in H$, we choose a uniform random bit $s^k_{\alpha_k}$ for each $k \in [\kappa]$.
    * We set $z_1 := (s^1_{\alpha_1} \oplus v_1[1], \ldots, s^{\kappa}_{\alpha_\kappa} \oplus v_1[\kappa]) \| z_1^{\mathsf{part}}$.
  - Skip the following changes if either of P_1 or P_2 is in H.
    * We obtain the transcript Z of Φ by implementing the messages sent by corrupt P_i for $i \in \{1, 2\}$ using $\mathsf{Faithful}(i, \{z_i\}_{i\in[2]}, \{b^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta})$ (where z_1 as extracted above and z_2 as received from P_2 in a round-2 broadcast).
    * Let st^*_T be the state at the end of the faithful execution of one of the corrupt parties (this value is the same for all corrupt parties). We send $(\mathsf{Corrupt}, \mathsf{sid}, \delta := \mathsf{st}^*_T[\ell/2])$ to the ideal functionality.
    * We receive $(\delta, \{z^i_\delta\}_{i\in[3,n]})$ from the ideal functionality.
    * For each $i \in [3, n]$ such that $P_i \in H$, we set $\mu^i := \{\mu^{i,k}_{\delta, z^i_\delta[k]}\}$ for each $k \in [\lambda]$.

- $\underline{\mathsf{Hybrid}_5 :}$ Skip this hybrid if $(P_1, P_2) \in H \times H$ or if $(P_1, P_2) \in C \times C$. In this hybrid, we make the following changes.
  - Initialize $\mathsf{aux} = \perp$.
  - **If $P_1 \in H$ and $P_2 \in C$.**
    * For each $k \in [\kappa]$, we intercept the message $(\mathsf{input}, k, P_2, (s^i_0, s^i_1))$ that $\mathcal{A}$ sends on behalf of corrupt P_2 to $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$.
    * For each $k \in [\kappa]$, we intercept the message $(\mathsf{predicate}, k, \mathsf{EQ}_{\beta_k})$ that $\mathcal{A}$ might send to the $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ functionality. If the number of such k for which $\mathcal{A}$ sends this message is greater than λ, then we send $(\mathsf{abort}, \mathsf{sid})$ to the functionality $\mathcal{F}_{\mathsf{dSel}}$.
    * On the other hand, if the number of such k for which $\mathcal{A}$ sends this message is less than or equal to λ, we do the following:
      · Let K be the subset of $[\kappa]$ such that $\mathcal{A}$ sends the message $(\mathsf{predicate}, k, \mathsf{EQ}_{\beta_k})$.
      · For every $k \in K$, we choose a uniform bit α_k.
      · If for any such k, $\alpha_k = \beta_k$ then, we send $(\mathsf{abort}, \mathsf{sid})$ to the functionality $\mathcal{F}_{\mathsf{dSel}}$.
      · Else, we set $\mathsf{aux} := (K, \{\alpha_k\}_{k\in K}, \{(s^i_0, s^i_1)\}_{i\in[\kappa]})$.
    * If we have not yet sent the abort message then for every $k \notin K$, we use the honest $P_1's$ randomness to choose a uniform bit α_k and set $z_1 := (s^1_{\alpha_1} \oplus v_1[1], \ldots, s^{\kappa}_{\alpha_\kappa} \oplus v_1[\kappa]) \| z_1^{\mathsf{part}}$.
  - **If $P_1 \in C$ and $P_2 \in H$.**
    * We set $\mathsf{aux} := \{(\alpha_k, s^k_{\alpha_k})\}_{k\in[\kappa]}$ where $s^k_{\alpha_k}$ was the bit that was randomly chosen while computing z_1.

- $\underline{\mathsf{Hybrid}_6 :}$ In this hybrid, we make the following changes:
  - If both P_1 and P_2 are honest, then compute z_1 using the honest P_1 and P_2's randomness.

– For each $i \in [2, n]$, if $P_i \in C$, we intercept the message $(\mathsf{input}, k, P_i, (sk_0^{i,k}, sk_1^{i,k}))$ that $\mathcal{A}$ sends to $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ for each $k \in [\kappa]$. For each $i \in [2, n]$, if $P_i \in H$, then we choose $sk_0^{i,k}, sk_1^{i,k}$ as the output of $\mathsf{Gen}(1^\lambda)$. We set $sk^{i,k} := sk^{i,k}_{z_1[k]}$. If $P_1 \in C$, for each $k \in [\kappa]$, we deliver $(\mathsf{output}, k, (z_1[k], \{sk^{i,k}\}_{i\in[2,n]})$ as the output from $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$.

– For each $i \in [2, n]$, if $P_i \in H$, we send $\{(\mathsf{Enc}(sk_0^{i,k}, \mathsf{lab}^{i,1}_{k,z_1[k]}), \mathsf{Enc}(sk_1^{i,k}, \mathsf{lab}^{i,1}_{k,z_1[k]}))\}_{k\in[\kappa]}$ in round-3.

- $\underline{\mathsf{Hybrid}_{6+t}}$ for $t \in [0, T]$. This distribution is the same as hybrid Hybrid_{6+t-1} except we change the distribution of the garbled circuits (in the third round) that play a role in the execution of the t^{th} round of the protocol Φ; namely, the action $\phi_t = (i^*, f, g, h)$. We describe the changes more formally below.

  – Skip the following change if both P_1 and P_2 are corrupted. In this hybrid, we complete the execution of Φ using honest party inputs and randomness. In this execution, the messages on behalf of corrupted parties are generated via faithful execution. Specifically, we send $\{z_i\}_{i\in\{P_1,P_2\}\cap C}$ to the honest parties on behalf of the corrupted party P_i in this mental execution of Φ. This starts the computation phase of Φ. In this computation phase, we generate the honest party messages using the inputs and random coins of the honest parties and generate the messages of the each malicious party P_i by executing $\mathsf{Faithful}\left(i, \{z_i\}_{i\in\{1,2\}}, \{b^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta}\right)$.

  – Let $\mathsf{Z} \in \{0, 1\}^T$ be the transcript obtained using the above step if either of P_1 or P_2 is honest. Otherwise, let Z be the transcript obtained as in Hybrid_4. Let st^*_T be the local state of one of the corrupted party the end of faithful execution and let st^*_t be the joint public state at the end of the t-th round of the computation phase. Finally, let $\alpha^* := \mathsf{st}^*_T[f]$, $\beta^* := \mathsf{st}^*_T[g]$ and $\gamma^* := \mathsf{st}^*_T[h]$. In Hybrid_{6+t} we make the following changes with respect to hybrid Hybrid_{6+t-1}:

    * We make the following two changes in how we generate messages for other honest parties P_i (i.e., $P_i \in H \setminus \{P_{i^*}\}$). We do not generate four $\mathsf{ots}^{i,t,\alpha,\beta}$ values but just one of them; namely, we generate $\mathsf{ots}^{i,t,\alpha^*,\beta^*}$ as $\mathsf{OT}_2(\mathsf{crs}^{i^*}, \mathsf{otr}^{i,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t})$. Second, we generate the garbled circuit

$$\left(\widetilde{C}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left(\mathsf{ots}^{i,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]\setminus\{h\}}\right)\right),$$

    where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{C}^{i,t+1}$ (for any $t + 1 \le T$) and for $t = T$, $\{\mathsf{lab}^{i,T+1}_{k,\mathsf{st}^*_T[k]}\}_{k\in[\ell]}$ are computed as per the protocol specification.

    * If $P_{i^*} \in C$ then skip these changes. We make two changes in how we generate messages on behalf of P_{i^*}. First, for all $\alpha, \beta \in \{0, 1\}$, we set $\mu^{i^*,t,\alpha^*,\beta^*}$ as $\mu^{i^*,t,\alpha^*,\beta^*}_{\mathsf{Z}_t}$ rather than $\mu^{i^*,t,\alpha^*,\beta^*}_{v_{i^*}[h]\oplus\mathsf{NAND}(v_{i^*}[f]\oplus\alpha^*, v_{i^*}[g]\oplus\beta^*)}$ (note that these two values are the same when using the honest party's input and randomness). Second, it generates the garbled circuit

$$\left(\widetilde{C}^{i^*,t}, \{\mathsf{lab}^{i^*,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left((\alpha^*, \beta^*, \gamma^*), \mu^{i^*,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i^*,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]}\right)\right),$$

where $\{\mathsf{lab}^{i^*,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{C}^{i^*,t+1}$ (for any $t+1 \le T$) and for $t = T$, $\{\mathsf{lab}^{i^*,T+1}_{k,\mathsf{st}^*_T[k]}\}_{k\in[\ell]}$ are computed as per the protocol specification.

- $\underline{\mathsf{Hybrid}_{7+T}}$: In this hybrid, for each $i \in [3,n]$ such that $P_i \in H$, we compute $(\widetilde{\mathsf{ChkC}}^i, \{\mathsf{lab}^{i,T+1}_k\}_{k\in[\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, 1^{|\mathsf{ChkC}^i|}, 1^\ell, \mathsf{out}_i)$ where out_i is $\perp$ is st^*_T leads to an abort of either P_1 or P_2 and is otherwise, equal to (z^i_δ, μ^i).

- $\underline{\mathsf{Hybrid}_{8+T}}$: In this hybrid, we modify the output phase of the computation to execute the garbled circuits provided by $\mathcal{A}$ on behalf of the corrupted parties and see if the execution of garbled circuits proceeds consistently with the transcript Z. If the computation succeeds then for each $P_i \in H$, we instruct the parties to output the result of the output computation phase; else, we instruct them to output $\perp$. This hybrid is computationally indistinguishable to the previous hybrid from the authenticity of input labels property of garbled circuits.

- $\underline{\mathsf{Hybrid}_{9+T}}$: Skip this hybrid change if P_1 and P_2 are in C. In this hybrid, we just change how the transcript Z, $\{z_i\}_{i\in H\cap\{P_1,P_2\}}$, and the value st^*_T are generated. Instead of generating these using honest party inputs in execution with a faithful execution of Φ, we generate it via the simulator Sim_Φ (of the maliciously secure protocol Φ) with aux as additional input. Specifically, we generate z_i as $(r_i \| 0^{\ell/2-(2\kappa+2)})$ where r_i is uniformly chosen random string of length $2\kappa+2$ for each $P_i \in H$ s.t. $i \in \{1,2\}$. To generate the transcript, we execute the simulator Sim_Φ where messages on behalf of each corrupted party P_i are generated using $\mathsf{Faithful}(i, \{z_i\}_{i\in[n]\setminus H}, \{b^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta})$. (Note that Sim_Φ might rewind $\mathsf{Faithful}$. This can be achieved since $\mathsf{Faithful}$ is just a polynomial time interactive procedure that can also be rewound.). Note that the value aux contains the inputs and the outputs of the adversary (corrupting either P_1 or P_2 in the protocol Φ) that is implicitly given to the $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$ functionality. It now follows from the statistical security of Φ that Hybrid_{8+T} is statistically close to Hybrid_{9+T}.

  We note that Hybrid_{9+T} is identically distributed to $\mathrm{EXEC}_{\mathcal{F}_{\mathsf{dSel}},\mathsf{Sim},\mathcal{Z}}$.

We now show that for each $i \in [9+T]$, either $\mathsf{Hybrid}_i \overset{c}{\approx} \mathsf{Hybrid}_{i-1}$, or $\mathsf{Hybrid}_i \overset{s}{\approx} \mathsf{Hybrid}_{i-1}$ or $\mathsf{Hybrid}_i \equiv \mathsf{Hybrid}_{i-1}$.

Claim 5.14. *Assuming the equivocal receiver security and the sender security of the oblivious transfer, we have* $\mathsf{Hybrid}_1 \overset{c}{\approx} \mathsf{Hybrid}_0$.

Proof. We consider a sequence of hybrids $\mathsf{Hybrid}_0 \equiv \mathsf{Hybrid}_{0,2}$ up to $\mathsf{Hybrid}_{0,n} \equiv \mathsf{Hybrid}_1$ where in $\mathsf{Hybrid}_{0,i}$ for $i \in [3,n]$, we change the distribution of crs^k and the first round messages generated by P_k (if $P_k \in H$) for every $k \le i$ as in Hybrid_1. To prove that $\mathsf{Hybrid}_0 \overset{c}{\approx} \mathsf{Hybrid}_1$, it is sufficient to show that for every $i \in [3,n]$, $\mathsf{Hybrid}_{0,i} \overset{c}{\approx} \mathsf{Hybrid}_{0,i-1}$. We now show this by considering the cases when P_i is honest or not.

- **Case-1:** $P_i \in C$. In the case $P_i \in C$, the only change in $\mathsf{Hybrid}_{0,i}$ and in $\mathsf{Hybrid}_{0,i-1}$ is in how crs^i is generated (the input extraction step does not affect the view of the adversary). Note that in $\mathsf{Hybrid}_{0,i-1}$, crs^i is generated as the output of $K_{\mathsf{OT}}(1^\lambda)$ whereas in $\mathsf{Hybrid}_{0,i}$, it is generated as the first component of the output of $\mathsf{Sim}_S^1(1^\lambda)$. Hence, it follows directly from the sender security of oblivious transfer that $\mathsf{Hybrid}_{0,i} \overset{c}{\approx} \mathsf{Hybrid}_{0,i-1}$.

- **Case-2:** $P_i \in H$. In order to show that $\mathsf{Hybrid}_{0,i-1} \overset{c}{\approx} \mathsf{Hybrid}_{0,i}$ when $P_i \in H$, we give a reduction to the equivocal receiver security of oblivious transfer. This reduction gives $\{z_b^i\}_{b\in\{0,1\}}$ as the challenge message and receives crs^i and $\{\mathsf{otm}_\gamma^{i,b}, \mu^{i,b}\}_{b\in\{0,1\}}$ from the challenger. It then uses the received values to generate the view of the adversary and compute the output as in $\mathsf{Hybrid}_{0,i-1}$. If the received messages crs^i and $\{\mathsf{otm}_\gamma^{i,b}, \mu^{i,b}\}_{b\in\{0,1\}}$ are generated using the real algorithms then the view of the adversary and the outputs of the honest parties are identical to the output of $\mathsf{Hybrid}_{0,i-1}$. Else, they are identical to the output of $\mathsf{Hybrid}_{0,i}$. This shows that any distinguisher against $\mathsf{Hybrid}_{0,i-1}$ and $\mathsf{Hybrid}_{0,i}$ can be used to break the equivocal receiver security of oblivious transfer which is a contradiction.

□

Claim 5.15. *Assuming the sender security of the oblivious transfer, we have* $\mathsf{Hybrid}_1 \overset{c}{\approx} \mathsf{Hybrid}_2$.

Proof. We consider a couple of intermediate distributions $\mathsf{Hybrid}_1 \equiv \mathsf{Hybrid}_{1,0}$, $\mathsf{Hybrid}_{1,1}$, and $\mathsf{Hybrid}_2 \equiv \mathsf{Hybrid}_{1,2}$. For each $i \in \{1,2\}$, in $\mathsf{Hybrid}_{1,i}$, we change the distribution of crs^i (if P_i is corrupted) and the second round OT messages generated by other honest parties with respect to the first round messages sent by P_i as in Hybrid_2. We now show that for each $i \in \{1,2\}$, $\mathsf{Hybrid}_{1,i} \overset{c}{\approx} \mathsf{Hybrid}_{1,i-1}$. We consider two cases depending on whether $P_i \in H$ or $P_i \in C$.

- **Case-1:** $P_i \in H$**.** Note that the only change in $\mathsf{Hybrid}_{1,i}$ and $\mathsf{Hybrid}_{1,i-1}$ is that in $\mathsf{Hybrid}_{1,i}$, for every $j \in [n]\backslash\{i\}$ such that $P_j \in H$, for every $t \in A_i$ and $\alpha, \beta \in \{0,1\}$, we compute $\mathsf{ots}^{j,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{crs}^i, \mathsf{otr}_1^{i,t,\alpha,\beta}, \mathsf{lab}^{j,t+1}_{h,b^{i,t,\alpha,\beta}}, \mathsf{lab}^{j,t+1}_{h,b^{i,t,\alpha,\beta}})$ (where $b^{i,t,\alpha,\beta} = v_i[h] \oplus \mathsf{NAND}(v_i[f] \oplus \alpha, v_i[g] \oplus \beta)$) whereas in $\mathsf{Hybrid}_{1,i-1}$, we compute $\mathsf{otm}^{j,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{crs}^i, \mathsf{otr}^{i,t,\alpha,\beta}, \mathsf{lab}^{j,t+1}_{h,0}, \mathsf{lab}^{j,t+1}_{h,1})$. Note that since $\mathsf{otr}^{i,t,\alpha,\beta}$ is computed by an honest party, it now follows from the semi-honest sender security of the oblivious transfer (which is implied by security against malicious receivers) that $\mathsf{Hybrid}_{1,i-1} \overset{c}{\approx} \mathsf{Hybrid}_{1,i}$.

- **Case-2:** $P_i \in C$**.** In this case, we give a reduction to the sender security of the oblivious transfer. Assume for the sake of contradiction that there exists a distinguisher D that can distinguish between the outputs of $\mathsf{Hybrid}_{1,i-1}$ and $\mathsf{Hybrid}_{1,i}$ with non-negligible advantage. We will use this distinguisher to construct an adversary $\mathcal{B}$ against the sender security of oblivious transfer.

  For each $t \in A_i$ such that $\phi_t = (i,f,g,h)$, $\mathcal{B}$ interacts with the challenger against the sender security and gives $\{\mathsf{lab}^{j,t+1}_{h,0}, \mathsf{lab}^{j,t+1}_{h,1}\}$ for each $j \in [n]$ such that $P_j \in H$ as the challenge sender strings. It receives crs^i from the external challenger and uses it to generate the crs. It then begins the interaction with $\mathcal{A}$. On receiving $\{\mathsf{otr}^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}$ from $\mathcal{A}$, $\mathcal{B}$ forwards $\mathsf{otr}^{i,t,\alpha,\beta}$ as the adversarial receiver message corresponding to the challenges $\{\mathsf{lab}^{j,t+1}_{h,0}, \mathsf{lab}^{j,t+1}_{h,1}\}$ for each $j \in [n]$ such that $P_j \in H$. It does this for each $t \in A_i, \alpha, \beta \in \{0,1\}$. $\mathcal{B}$ receives $\{\mathsf{ots}^{j,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}$ for each $P_j \in H$ from the external challenger. $\mathcal{B}$ uses these strings to generate the third round message of the protocol and computes the output of all honest parties as in $\mathsf{Hybrid}_{1,i-1}$. It finally runs D on the view of the adversary and the outputs of the honest parties and outputs whatever D outputs.

Now, if the distribution of crs^i and the second round OT messages generated by the challenger are computed using the real algorithms, then the input to D is identical to $\mathsf{Hybrid}_{1,i-1}$. Else, it is identical to $\mathsf{Hybrid}_{1,i}$. Thus, if D can distinguish between $\mathsf{Hybrid}_{1,i-1}$ and $\mathsf{Hybrid}_{1,i}$ with non-negligible advantage, then $\mathcal{B}$ can break the sender security of the oblivious transfer with the same advantage which is a contradiction.

□

Claim 5.16. *Assuming the equivocal receiver security of the oblivious transfer,* $\mathsf{Hybrid}_2 \overset{c}{\approx} \mathsf{Hybrid}_3$.

Proof. We consider a sequence of hybrids $\mathsf{Hybrid}_2 \equiv \mathsf{Hybrid}_{2,0}$ up to $\mathsf{Hybrid}_{2,2} \equiv \mathsf{Hybrid}_3$ where in $\mathsf{Hybrid}_{2,i}$ for $i \in \{1,2\}$, we change the distribution of crs^k and the first round messages generated by P_k (if $P_k \in H$) for every $k \leq i$. To prove that $\mathsf{Hybrid}_2 \overset{c}{\approx} \mathsf{Hybrid}_3$, it is sufficient to show that for every $i \in \{1,2\}$, $\mathsf{Hybrid}_{2,i} \overset{c}{\approx} \mathsf{Hybrid}_{2,i-1}$.

To show that $\mathsf{Hybrid}_{2,i-1} \overset{c}{\approx} \mathsf{Hybrid}_{2,i}$ when $P_i \in H$, we give a reduction to the equivocal receiver security of oblivious transfer. This reduction gives $\{v_i[h] \oplus \mathsf{NAND}(v_i[f] \oplus \alpha, v_i[g] \oplus \beta)\}_{t \in A_i, (\alpha,\beta) \in \{0,1\} \times \{0,1\}, \phi_t = (i,f,g,h)}$ as the challenge message bits and receives crs^i and $\{\mathsf{otr}^{i,t,\alpha,\beta}, \mu^{i,t,\alpha,\beta}\}_{t \in A_i, (\alpha,\beta) \in \{0,1\} \times \{0,1\}}$ from the challenger. It then uses the received values to generate the view of the adversary and compute the output of honest parties as in $\mathsf{Hybrid}_{2,i-1}$. If the received messages crs^i and $\{\mathsf{otr}^{i,t,\alpha,\beta}, \mu^{i,t,\alpha,\beta}\}_{t \in A_i, (\alpha,\beta) \in \{0,1\} \times \{0,1\}}$ are generated using the real algorithms then the view of the adversary and the outputs of the honest parties are identical to the output of $\mathsf{Hybrid}_{2,i-1}$. Else, they are identical to the output of $\mathsf{Hybrid}_{2,i}$. This shows that any distinguisher against $\mathsf{Hybrid}_{2,i-1}$ and $\mathsf{Hybrid}_{2,i}$ can be used to break the equivocal receiver security of oblivious transfer. □

Claim 5.17. $\mathsf{Hybrid}_3 \equiv \mathsf{Hybrid}_4$

Proof. Note that for each $k \in [\kappa]$, $z_1[k]$ computed in this step is identical to $x_1[k] \oplus v_1[k]$ that will be output by $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ to a corrupt P_1. Thus, we observe that the $\mathsf{faithful}$ procedure perfectly emulates the messages that $\mathcal{A}$ sends on behalf of the corrupt parties in the conforming protocol Φ. Thus, for each $i \in [3,n]$ such that $P_i \in H$, $z^i_{\mathsf{st}^*[\ell/2]}$ denotes the correct output obtained by the parties in the case when P_1 and P_2 are corrupt. Additionally, we haven't changed the view of the adversary between Hybrid_3 and Hybrid_4 and thus, these two hybrids are identical. □

Claim 5.18. $\mathsf{Hybrid}_4 \overset{s}{\approx} \mathsf{Hybrid}_5$

Proof. Note that the only difference between Hybrid_5 and Hybrid_4 is that in Hybrid_5, if the number of indices for which adversary $\mathcal{A}$ sends the message $(\mathsf{predicate}, \cdot, \cdot)$ to the $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ functionality is greater than λ then we abort.

Let $K \subseteq [\kappa]$ be the indices k such that $\mathcal{A}$ sends $(\mathsf{predicate}, k, \mathsf{EQ}_{\beta_k})$ to $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ functionality. We argue that if $|K| > \lambda$ then an honest party P_1 outputs abort except with probability at most $2^{-\lambda}$. This follows from the fact that $\alpha_1, \ldots, \alpha_\kappa$ are chosen uniformly at random and hence, for any k, the probability that $\beta_k = \alpha_k$ is $1/2$. Hence, the probability for every $k \in K$, $\beta_k \neq \alpha_k$ is at most $2^{-\lambda}$. □

Claim 5.19. *Assuming the semantic security of the symmetric key encryption, we have* $\mathsf{Hybrid}_5 \overset{c}{\approx} \mathsf{Hybrid}_6$.

Proof. Assume for the sake of contradiction that there exists a distinguisher D that can distinguish between the Hybrid_5 and Hybrid_6 with non-negligible advantage. We now use D to construct an adversary $\mathcal{B}$ that breaks the semantic security of the symmetric key encryption.

By a standard averaging argument, we infer that there exists $i \in [2, n]$ such that $P_i \in H$ and two distributions Hybrid'_0 and Hybrid'_1 (described below) such that D can distinguish between Hybrid'_0 and Hybrid'_1 with non-negligible advantage. In both these distributions, for any $i' \in [2, n]$ such that $P_{i'} \in H$ and $i' < i$, the ciphertexts generated by $P_{i'}$ in the third round are identical to its distribution in Hybrid_6, whereas for any $i' > i$, these ciphertexts are identical to its distribution in Hybrid_5. The only difference between these two hybrids is in the distribution of the ciphertexts sent by P_i in the last round. In Hybrid'_0, it is distributed as in Hybrid_5, whereas in Hybrid'_1, it is identical to Hybrid_6.

For each $k \in [\kappa]$, $\mathcal{B}$ interacts with κ challenge oracles (each instantiated with an independent secret key). It gives $\{\mathsf{lab}^{i,1}_{1-z_1[k]}, \mathsf{lab}^{i,1}_{z_1[k]}\}$ as the two challenge messages to the k-th oracle for each $k \in [\kappa]$. It receives $\mathsf{ct}^*_{i,k}$ for each $k \in [\kappa]$. $\mathcal{B}$ chooses an independent key $sk^{i,k}_{z_1[k]}$ and generates $\mathsf{ct}^{i,k}_{z_1[k]} = \mathsf{Enc}(sk^{i,k}_{z_1[k]}, \mathsf{lab}^{i,1}_{k,z_1[k]})$ and sets $\mathsf{ct}^{i,k}_{1-z_1[k]} := \mathsf{ct}^*_{i,k}$. It sends $\{(\mathsf{ct}^{i,k}_0, \mathsf{ct}^{i,k}_1)\}_{k\in[\kappa]}$ as the ciphertexts from party P_i in the final round. It generates the rest of the messages and the output of the honest parties as Hybrid'_0. It finally runs the distinguisher on the view of the adversary and the outputs of the honest parties and outputs whatever D outputs.

Note that if $\mathsf{ct}^*_{i,k}$ is an encryption of $\mathsf{lab}^{i,1}_{1-z_1[k]}$ then the inputs to D are identical to Hybrid'_0. Otherwise, the inputs to D are identical to Hybrid'_1. This contradicts the semantic security of the symmetric key encryption. □

Claim 5.20. *Assuming the security of garbled circuits, we have for each $t \in [T]$ that $\mathsf{Hybrid}_{6+t} \overset{c}{\approx} \mathsf{Hybrid}_{6+t-1}$.*

Proof. Let $\mathsf{Z} \in \{0,1\}^T$ be the transcript obtained as in the hybrid description if either of P_1 or P_2 is honest. Otherwise, let Z be the transcript obtained as in Hybrid_4. Let st^*_T be the joint public state at the end of faithful execution and let st^*_t be the joint public state at the end of the t-th round of the computation phase. Let $\phi_t = (i^*, f, g, h)$. Finally, let $\alpha^* := \mathsf{st}^*_T[f]$, $\beta^* := \mathsf{st}^*_T[g]$ and $\gamma^* := \mathsf{st}^*_T[h]$. To show that $\mathsf{Hybrid}_{6+t-1} \overset{c}{\approx} \mathsf{Hybrid}_{6+t}$, we consider a couple of intermediate distributions:

- $\mathsf{Hybrid}_{6+t-1,1}$: We make the following two changes in how we generate messages for other honest parties P_i (i.e., $P_i \in H \setminus \{P_{i^*}\}$). We do not generate four $\mathsf{ots}^{i,t,\alpha,\beta}$ values but just one of them; namely, we generate $\mathsf{ots}^{i,t,\alpha^*,\beta^*}$ as $\mathsf{OT}_2(\mathsf{crs}^{i^*}, \mathsf{otr}^{i,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t})$ (note that if i^* is honest then $\mathsf{Z}_t = v_{i^*}[h] \oplus \mathsf{NAND}(v_{i^*}[f^*] \oplus \alpha^*, v_{i^*}[g] \oplus \beta^*)$). Second, we generate the garbled circuit

$$\left(\widetilde{C}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left(\mathsf{ots}^{i,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]\setminus\{h\}}\right)\right),$$

where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{C}^{i,t+1}$ (for any $t+1 \le T$) and for $t = T$, $\{\mathsf{lab}^{i,T+1}_{k,\mathsf{st}^*_T[k]}\}_{k\in[\ell]}$ are computed as per the protocol specification.

It follows from $|H \setminus \{P_{i^*}\}|$ invocations of the security of garbled circuits that $\mathsf{Hybrid}_{6+t-1,1} \overset{c}{\approx} \mathsf{Hybrid}_{6+t-1}$.

- $\mathsf{Hybrid}_{6+t-1,2}$: Skip this hybrid change if $P_{i^*} \notin H$. We set $\mu^{i^*,t,\alpha^*,\beta^*}$ as $\mu^{i^*,t,\alpha^*,\beta^*}_{v_{i^*}[h]\oplus\mathsf{NAND}(v_{i^*}[f]\oplus\alpha^*,v_{i^*}[g]\oplus\beta^*)}$ and compute

$$\left(\widetilde{C}^{i^*,t}, \{\mathsf{lab}_k^{i^*,t}\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left((\alpha^*,\beta^*,\gamma^*), \mu^{i^*,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i^*,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]}\right)\right),$$

  where $\{\mathsf{lab}^{i^*,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{C}^{i^*,t+1}$ (for any $t+1 \le T$) and for $t = T$, $\{\mathsf{lab}^{i^*,T+1}_{k,\mathsf{st}^*_T[k]}\}_{k\in[\ell]}$ are computed as per the protocol specification.

  It follows from the security of the garbled circuit that $\mathsf{Hybrid}_{6+t-1,2} \overset{c}{\approx} \mathsf{Hybrid}_{6+t-1,1}$.

- $\mathsf{Hybrid}_{6+t-1,3}$: Skip this hybrid change if $P_{i^*} \notin H$. In this hybrid, we set $\mu^{i^*,t,\alpha^*,\beta^*}$ as $\mu^{t,\alpha^*,\beta^*}_{\mathsf{Z}_t}$ rather than $\mu^{i^*,t,\alpha^*,\beta^*}_{v_{i^*}[h]\oplus\mathsf{NAND}(v_{i^*}[f]\oplus\alpha^*,v_{i^*}[g]\oplus\beta^*)}$. Since for the case of honest parties, these two values are the same, it follows that $\mathsf{Hybrid}_{6+t-1,3}$ is identically distributed to $\mathsf{Hybrid}_{6+t-1,2}$.

  Observe that $\mathsf{Hybrid}_{6+t-1,3}$ is identically distributed to Hybrid_{6+t}.

□

Claim 5.21. *Assuming the security of garbled circuits, we have* $\mathsf{Hybrid}_{6+T} \overset{c}{\approx} \mathsf{Hybrid}_{7+T}$.

Proof. Notice that the only difference between Hybrid_{6+T} and Hybrid_{7+T} is that in Hybrid_{7+T}, all the garbled circuits $\widetilde{\mathsf{ChkC}}^i$ for every $i \in [3,n]$ such that $P_i \in H$ is generated using $\mathsf{Sim}_{\mathsf{GC}}$ whereas in Hybrid_{6+T}, it is generated using the real garbling procedure. It directly follows from $|H \cap \{P_3,\ldots,P_n\}|$ invocations of the security of garbled circuits that $\mathsf{Hybrid}_{6+T} \overset{c}{\approx} \mathsf{Hybrid}_{7+T}$. □

Since we argued inline that $\mathsf{Hybrid}_{7+T} \overset{c}{\approx} \mathsf{Hybrid}_{8+T}$ and $\mathsf{Hybrid}_{8+T} \overset{s}{\approx} \mathsf{Hybrid}_{9+T}$, this completes the proof of the lemma.

□

5.4 Third Step: Bootstrapping from Special to General Functions in 3 Rounds

In this section, we build a 3-round MPC protocol for any multiparty function f in the $\mathcal{F}_{\mathsf{dSel}}$-hybrid model. The main theorem we show in this subsection is the following.

Theorem 5.22. *Let f be a n-party functionality. There exists a protocol Π_f that UC-realizes f in three rounds against malicious adversaries corrupting an arbitrary number of parties. Π_f makes black-box use of a two-round, malicious-secure OT with equivocal receiver security and is in the $\mathcal{F}_{\mathsf{dSel}}$-hybrid model.*

Building Π_f. The protocol Π_f is obtained as a result of applying the round-collapsing compiler in [GS18, GIS18] to perfect/statistical protocols in the OT-correlations model (e.g., [Kil88, IPS08]). Specifically, the protocol we round-collapse has the following structure.

- **Generating OT Correlations.** Every pair of parties invoke a certain number of OT executions on uniformly chosen random inputs.

- **Protocol** Π**.** The parties augment their inputs with the OT correlations generated in the previous phase. The parties then use the perfect/statistical protocol from [Kil88, IPS08] in the OT correlations model to securely compute f.

Let Φ be the conforming protocol obtained as a result of the transformation in Theorem 5.10 to Π. For every $i, j \in [n]$ such that $i \neq j$, let κ be the number of random OT correlations required between party P_i (acting as the receiver) and P_j (acting as the sender) in the protocol Φ. The building blocks we use for Π_f are the conforming protocol Φ, a two-round, malicious-secure OT with equivocal receiver security, a garbling scheme for circuits and a symmetric key encryption. Further, we assume without loss of generality, that the first $(n-1)\kappa$ bits of the augmented input of party P_i in Φ contains the bits obtained from every other party (acting as sender) in the OT correlations generation phase. Specifically, the first κ bits are the received bits from P_1 (if $i \neq 1$) and the second set of κ bits are the received bits from P_2 (if $i \neq 2$) and so on. We denote a function $\mathsf{GetIndex}$ that takes i, j, k as inputs (where $i, j \in [n]$, $i \neq j$ and $k \in [\kappa]$) and returns an index $\mathsf{ind} \in [\ell]$ of the state st of the conforming protocol that corresponds to the received bit in the k-th OT correlation between P_i (acting as the receiver) and P_j (acting as the sender). We now present an information description of Π_f below and the formal description in Figure 10.

Building on the round-collapsing compiler of [GS18, GIS18], the main challenge in Π_f is in making the first set of labels for the joint state available within 3 rounds. Unlike [GS18, GIS18], the input to the conforming protocol in our case not only includes the actual inputs of the parties, but also the OT correlations. The generation of the latter (to be specific, the output bit of an OT) is completed only at the end of round-2. As a result, the public state of a party can be made available to all only in round-3 and the labels for the joint state in round-4. We overcome this challenge using the double selection $\mathcal{F}_{\mathsf{dSel}}$ functionality. The double selection functionality allows the parties to learn the labels corresponding to masked value of the correlation bits at the end of round-3 allowing them to trigger the evaluation of garbled circuits at the end of round-3.

Protocol Π_f

Inputs: P_i for $i \in [n]$ inputs x_i.

Output: Every party outputs $f(x_1, \ldots, x_n)$.

Primitives and Functionalities: (a) A malicious-secure two-round OT with equivocal receiver security $(K_{\mathsf{OT}}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ (see Section 3.3), (b) Functionality $\mathcal{F}_{\mathsf{dSel}}$ (c) The conforming protocol Φ obtained as a result of the transformation in Theorem 5.10 to Π as discussed (c) Garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$ (see Section 3.2) (d) A symmetric-key Encryption Scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

Common Random/Reference String: For each $i \in [n]$, sample $\mathsf{crs}^i \leftarrow K_{\mathsf{OT}}(1^\lambda)$ and output $\{\mathsf{crs}^i\}_{i \in [n]}$ as the common random/reference string.

Round-1: In the first round,

- Each P_i runs $\mathsf{pre}(1^\lambda, i)$ to get v_i.
- For each $i, j \in [n]$ and $i \neq j$ and for each $k \in [\kappa]$, the parties invoke an instance of functionality $\mathcal{F}_{\mathsf{dSel}}$ as follows:
  - P_i, taking the role of P_1, sends $(\mathsf{input}, (i, j, k), P_i, (\alpha_k^{i,j}, r_k^{i,j}))$ to $\mathcal{F}_{\mathsf{dSel}}$ where $\alpha_k^{i,j}$ is a uniformly chosen bit and $r_k^{i,j} := v_i[\mathsf{GetIndex}(i, j, k)]$.

- P_j, taking the role of P_2, sends $(\mathsf{input}, (i,j,k), P_j, (y^{i,j}_{k,0}, y^{i,j}_{k,1})$ to $\mathcal{F}_{\mathsf{dSel}}$ where $y^{i,j}_{k,0}, y^{i,j}_{k,1}$ are uniformly chosen bits.
- For every $s \in [n]$, P_s inputs $(\mathsf{input}, (i,j,k), P_s, (sk^{s,i,j}_{k,0}, sk^{s,i,j}_{k,1}))$ to $\mathcal{F}_{\mathsf{dSel}}$ where $sk^{s,i,j}_{k,0}, sk^{s,i,j}_{k,1}$ are sampled using $\mathsf{Gen}(1^\lambda)$.

Round-2: In the second round, every P_i does the following

- It sets $x_i^{\mathsf{part}} := (x_i, \{\alpha^{i,j}_k, y^{j,i}_{k,0}, y^{j,i}_{k,1}\}_{j\in[n]\setminus\{i\}, k\in[\kappa]})$.
- It sets $z_i^{\mathsf{part}} := x_i^{\mathsf{part}} \oplus v_i[(i-1)\ell/n + (n-1)\kappa + 1, i\ell/n]$.
- For each $i \in [n]$ and for each t such that $\phi_t = (i, f, g, h)$ (A_i is the set of such values of t), for each $\alpha, \beta \in \{0,1\}$, it computes: $(\mathsf{otr}^{i,t,\alpha,\beta}, \mu^{i,t,\alpha,\beta}) \leftarrow \mathsf{OT}_1(\mathsf{crs}^i, v_i[h] \oplus \mathsf{NAND}(v_i[f]\oplus\alpha, v_i[g]\oplus\beta))$.
- It broadcasts $\left(z_i^{\mathsf{part}}, \{\mathsf{otr}^{i,t,\alpha,\beta}\}_{t\in A_i, \alpha,\beta\in\{0,1\}}\right)$.

Round-3: In the final round, each party P_i does the following:

- It sets $\mathsf{st} = \left((0^{(n-1)\kappa} \| z_1^{\mathsf{part}}) || \ldots || (0^{(n-1)\kappa} \| z_n^{\mathsf{part}})\right)$.
- It sets $\mathsf{lab}^{i,T+1} := \{\mathsf{lab}^{i,T+1}_{k,0}, \mathsf{lab}^{i,T+1}_{k,1}\}_{k\in[\ell]}$ where for each $k \in [\ell]$ and $b \in \{0,1\}$, $\mathsf{lab}^{i,T+1}_{k,b} := \bot$.
- **for** each t from T down to 1,
  1. Let ϕ_t as (i^*, f, g, h).
  2. If $i = i^*$, then it computes $\left(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}\right) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \{\mu^{i,t,\alpha,\beta}\}_{\alpha,\beta}, \bot, \mathsf{lab}^{i,t+1}])$ (where $C^{i,t}$ is described in Figure 9).
  3. If $i \neq i^*$ then for every $\alpha, \beta \in \{0,1\}$, it sets $\mathsf{ots}^{i^*,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{crs}^{i^*}, \mathsf{otr}^{i^*,t,\alpha,\beta}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$ and computes $\left(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}\right) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \bot, \{\mathsf{ots}^{i,t,\alpha,\beta}\}_{\alpha,\beta}, \mathsf{lab}^{i,t+1}])$ (where $C^{i,t}$ is described in Figure 9).
- Each P_i broadcasts $\{\widetilde{C}^{i,t}\}_{t\in[T]}$, and for each $j \in [n]$ and $k \notin [(j-1)\ell/n+1, (j-1)\ell/n+(n-1)\kappa]$, P_i broadcasts $\mathsf{lab}^{i,1}_{k,\mathsf{st}[k]}$. In addition, P_i broadcasts for each $j, j' \in [n]$ such that $j \neq j'$ and $k \in [\kappa]$, $\left(\mathsf{ct}^{i,j,j'}_{k,0} = \mathsf{Enc}(sk^{i,j,j'}_{k,0}, \mathsf{lab}^{i,1}_{\mathsf{GetIndex}(j,j',k),0}), \mathsf{ct}^{i,j,j'}_{k,1} = \mathsf{Enc}(sk^{i,j,j'}_{k,1}, \mathsf{lab}^{i,1}_{\mathsf{GetIndex}(j,j',k),1})\right)$.

Output: Each party P_i does the following:

- For each $j, j' \in [n]$ such that $j \neq j'$ and for each $k \in [\kappa]$, let $\eta := \mathsf{GetIndex}(i,j,k)$ and do the following:
  1. Receive $(\mathsf{output}, (j,j',k), P_i, (z_\eta, \{sk^{s,j,j'}_{k,z_\eta}\}_{s\in[n]}))$ from $\mathcal{F}_{\mathsf{dSel}}$ functionality.
  2. Reset $\mathsf{st}[\eta] = z_\eta$.
  3. For each $s \in [n]$, set $\mathsf{lab}^{s,1}_{\eta,\mathsf{st}[\eta]} \leftarrow \mathsf{Dec}(sk^{s,j,j'}_{k,\mathsf{st}[\eta]}, \mathsf{ct}^{s,j,j'}_{k,\mathsf{st}[\eta]})$.
- For every $j \in [n]$, let $\widetilde{\mathsf{lab}}^{j,1} = \{\mathsf{lab}^{j,1}_{k,\mathsf{st}[k]}\}_{k\in[\ell]}$, where $\{\mathsf{lab}^{j,1}_{k,\mathsf{st}[k]}\}_{k\in[(j-1)\ell/n+1,(j-1)\ell/n+(n-1)\kappa]}$ are decrypted as above and the rest received from P_j's round-3 message.
- **for** each t from 1 to T do:
  1. Parse ϕ_t as (i^*, f, g, h).
  2. Compute $((\alpha,\beta,\gamma), \mu, \widetilde{\mathsf{lab}}^{i^*,t+1}) := \mathsf{Eval}(\widetilde{C}^{i^*,t}, \widetilde{\mathsf{lab}}^{i^*,t})$.
  3. Set $\mathsf{st}[h] := \gamma$.
  4. **for** each $j \neq i^*$ do:

(a) Compute $(\mathsf{ots}, \{\mathsf{lab}^{j,t+1}_{k,\mathsf{st}[k]}\}_{k\in[\ell]\setminus\{h\}}) := \mathsf{Eval}(\widetilde{C}^{j,t}, \widetilde{\mathsf{lab}}^{j,t})$.

(b) Recover $\mathsf{lab}^{j,t+1}_{h,\mathsf{st}[h]} := \mathsf{OT}_3(\mathsf{crs}^{i^*}, \mathsf{ots}, (\gamma, \mu))$.

(c) Set $\widetilde{\mathsf{lab}}^{j,t+1} := \{\mathsf{lab}^{j,t+1}_{k,\mathsf{st}[k]}\}_{k\in[\ell]}$.

- Output $\mathsf{post}(\mathsf{st}, v_i)$.

Figure 10: Protocol Π_f

Lemma 5.23. *Let $\mathcal{A}$ be an (possibly malicious) adversary corrupting an arbitrary subset of parties in the protocol Π_f. There exists a simulator* Sim *such that for any environment $\mathcal{Z}$,*

$$\mathrm{EXEC}_{\mathcal{F}_f,\mathsf{Sim},\mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}_{\Pi_f,\mathcal{A},\mathcal{Z}}$$

Proof. Let $C \subset \{P_1, \ldots, P_n\}$ be the set of parties corrupted by $\mathcal{A}$ and let $H = \{P_1, \ldots, P_n\} \setminus C$ denote the set of uncorrupted parties. Since we assume that $\mathcal{A}$ is static, the set of corrupted parties C is decided before the beginning of the protocol. We now give the description of the ideal world simulator Sim. Sim internally uses the simulators $(\mathsf{Sim}_R, \mathsf{Sim}_S)$ of the oblivious transfer (see Section 3.3), Sim_Φ of the conforming protocol Φ, and the simulator for garbled circuit $\mathsf{Sim}_{\mathsf{GC}}$.

Interaction with environment $\mathcal{Z}$. For every input value corresponding to the set of corrupted parties C that Sim receives from the environment $\mathcal{Z}$, Sim writes this value to $\mathcal{A}$'s input tape. Similarly, the contents of $\mathcal{A}$'s output tape is written to Sim's output tape. We now describe how Sim simulates the interaction of honest parties with $\mathcal{A}$.

Simulating the interaction with $\mathcal{A}$: For every concurrent interaction with the session identifier sid that $\mathcal{A}$ may start, the simulator does the following:

Common Random/Reference String Generation: Sim generates the crs as follows:

- For each $i \in [n]$, if $P_i \in C$, then Sim samples $\underline{(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}^1_S(1^\lambda)}$. Else, it samples $\underline{(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}^1_R(1^\lambda)}$.
- Sim sets the crs to $(\mathsf{crs}^1, \ldots, \mathsf{crs}^n)$.

Initialization and Round-1. Sim does the following:

- Initialize $\mathsf{aux} = \perp$. aux is a list contains the input and output of every corrupt party given to each invocation of the OT with an honest party in the correlation generation phase.
- **Updating aux.**
  - For each $P_i \in H, P_j \in C$ and for each $k \in [\kappa]$, it adds $((i,j,k),(y^{i,j}_{k,0}, y^{i,j}_{k,1}))$ to aux from the intercepted message $(\mathsf{input}, (i,j,k), P_j, (y^{i,j}_{k,0}, y^{i,j}_{k,1}))$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{dSel}}$ on behalf of corrupt P_j.

– For each $P_i \in C, P_j \in H$ and for each $k \in [\kappa]$, it adds $((i,j,k),(\alpha_k^{i,j}, y^{i,j}_{k,\alpha_k^{i,j}}))$ where $y^{i,j}_{k,\alpha_k^{i,j}}$ is uniformly chosen to aux from the intercepted message $(\mathsf{input},(i,j,k),P_i,(\alpha_k^{i,j}, r_k^{i,j}))$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{dSel}}$ on behalf of corrupt P_i.

- For each $i \in [n]$ such that $P_i \in H$, Sim does the following:
  – It sets $z_i := r_i \| 0^{\ell/n-m}$ where r_i is chosen uniformly from $\{0,1\}^m$ where m is the total length of the inputs of P_i (which includes the actual input and OT correlation).
  – It sets $z_i^{\mathsf{part}} = z_i[(n-1)\kappa+1, \ell/n]$.

Round-2 message from Sim to $\mathcal{A}$. For each $i \in [n]$ such that $P_i \in H$, Sim does the following:

- For each $t \in A_i$ and for each $\alpha,\beta \in \{0,1\}$, it generates $(\mathsf{otr}^{i,t,\alpha,\beta}, \mu_0^{i,t,\alpha,\beta}, \mu_1^{i,t,\alpha,\beta}) \leftarrow \mathsf{Sim}^2_R(\mathsf{crs}^i, \mathsf{td}^i)$.
- Sim sends the second round message on behalf of the honest parties to $\mathcal{A}$.
- **Extraction from OT.** For each $i \in [n]$ such that $P_i \in C$ and for each $t \in A_i$, $\alpha,\beta \in \{0,1\}$, Sim computes $b^{i,t,\alpha,\beta} = \mathsf{Sim}^2_S(\mathsf{crs}^i, \mathsf{td}^i, \mathsf{otr}^{i,t,\alpha,\beta})$.
- **Setting up st^*.**

  Initialization: It initializes $\mathsf{st}^* = \Big((0^{(n-1)\kappa} \| z_1^{\mathsf{part}})|| \ldots ||(0^{(n-1)\kappa} \| z_n^{\mathsf{part}})\Big)$.

  Setting the positions for OT output Correlation for Honest parties: For each $j \in [n]$ such that $P_j \in H$ and for each $k \in [(j-1)\ell/n+1, (j-1)\ell/n+(n-1)\kappa]$, it sets $\mathsf{st}^*[k] = z_j[k-(j-1)\ell/n]$.

  Setting the input positions of every $P_i \in C$: For every $i \in [n]$ such that $P_i \in C$:

  For every $j \neq i$ such that $P_j \in H$ and for each $k \in [\kappa]$, let $(\mathsf{input},(i,j,k),P_i,(\alpha_k^{i,j}, r_k^{i,j}))$ be the intercepted message that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{dSel}}$.
  – It sets $\mathsf{st}^*[\mathsf{GetIndex}(i,j,k)] := y^{i,j}_{k,\alpha_k^{i,j}} \oplus r_k^{i,j}$ where $y^{i,j}_{k,\alpha_k^{i,j}}$ was the bit that Sim chose previously.

  For every $j \neq i$ such that $P_j \in C$ and for each $k \in [\kappa]$, let $(\mathsf{Corrupt},(i,j,k),\beta_k^{i,j})$ be the intercepted message that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{dSel}}$
  – It sets $\mathsf{st}^*[\mathsf{GetIndex}(i,j,k)] := \beta_k^{i,j}$.

Faithful execution. We define an interactive procedure $\mathsf{Faithful}(i, \mathsf{st}^*, \{b^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta})$ that on input $i \in [n]$, st^*, $\{b^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}$ produces protocol Φ message on behalf of party P_i (acting consistently/faithfully with the extracted values) as follows: For $t \in \{1 \cdots T\}$

- Parse $\phi_t = (i^*, f, g, h)$.
- If $i \neq i^*$ then it waits for a bit from P_{i^*} and sets $\mathsf{st}^*[h]$ to be the received bit once it is received. Otherwise, set $\mathsf{st}^*[h] := b^{i^*,t,\mathsf{st}^*[f],\mathsf{st}^*[g]}$ and send it to all the other parties.

Round-3 message from Sim to $\mathcal{A}$. To generate the round-3 message Sim does the following:

- Sim initializes Sim_Φ with value $(H, \mathsf{st}^*, \mathsf{aux})$. This starts the computation phase of Φ with the simulator Sim_Φ.

- Sim provides computation phase messages from corrupted parties to Sim_Φ by following a faithful execution. More formally, for every $P_i \in C$ where $i \in [n]$, Sim generates messages on behalf of P_i for Sim_Φ using the procedure $\mathsf{Faithful}(i, \mathsf{st}^*, \{b^{i,t,\alpha,\beta}\}_{t \in A_i, \alpha, \beta})$.

- At some point during the execution, Sim_Φ will return the extracted inputs $\{x_i\}_{P_i \in C}$ of the corrupted parties. For each $P_i \in C$, Sim sends $(\mathsf{input}, \mathsf{sid}, P_i, x_i)$ to the ideal functionality $\mathcal{F}_f$ and obtains the output out. It sends out as the output to Sim_Φ. Sim_Φ completes the rest of the execution of the protocol.

- Let $\mathsf{Z} \in \{0,1\}^t$ where Z_t is the bit sent in the t^{th} round of the computation phase of Φ. And let st_T^* be the state value at the end of faithful execution of one of the corrupted parties (this value is the same for all the parties). For each $t \in \cup_{i \in H} A_i$ and $\alpha, \beta \in \{0,1\}$, set $\mu^{i,t,\alpha,\beta} := \mu_{\mathsf{Z}_t}^{i,t,\alpha,\beta}$.

- For each $i, j \in [n]$ s.t. $i \neq j$ and for each $k \in [\kappa]$, Sim does the following:
  - Let $\eta := \mathsf{GetIndex}(i, j, k)$.
  - For every $s \in [n]$ such that $P_s \in H$, it samples $sk_{k,0}^{s,i,j}, sk_{k,1}^{s,i,j}$ using $\mathsf{Gen}(1^\lambda)$.
  - For every $s \in [n]$ such that $P_s \in C$, it intercepts the message $(\mathsf{input}, (i,j,k), P_s, (sk_{k,0}^{s,i,j}, sk_{k,1}^{s,i,j}))$ that $\mathcal{A}$ sends on behalf of corrupt P_s.
  - It delivers $(\mathsf{output}, (i,j,k), (\mathsf{st}^*[\eta], \{sk_{k,\mathsf{st}^*[\eta]}^{s,i,j}\}_{s \in [n]})$ as the output from $\mathcal{F}_{\mathsf{dSel}}$ to $\mathcal{A}$.

- For each $i \in [n]$ such that $P_i \in H$, Sim does the following:
  - For each $k \in [\ell]$, it sets $\mathsf{lab}_k^{i,T+1} := \bot$.
  - **for** each t from T down to 1,
    - Parse ϕ_t as (i^*, f, g, h).
    - Set $\alpha^* := \mathsf{st}_T^*[f]$, $\beta^* := \mathsf{st}_T^*[g]$, and $\gamma^* := \mathsf{st}_T^*[h]$.
    - If $i = i^*$ then compute
    $$\left(\widetilde{C}^{i,t}, \{\mathsf{lab}_{k,\mathsf{st}_T^*[k]}^{i,t}\}_{k \in [\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left((\alpha^*, \beta^*, \gamma^*), \mu^{i^*,t,\alpha^*,\beta^*}, \{\mathsf{lab}_k^{i,t+1}\}_{k \in [\ell]}\right)\right).$$
    - If $i \neq i^*$ then set $\mathsf{ots}^{i,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2(\mathsf{crs}^{i^*}, \mathsf{otr}^{i^*,t,\alpha^*,\beta^*}, \mathsf{lab}_h^{i,t+1}, \mathsf{lab}_h^{i,t+1})$ and compute
    $$\left(\widetilde{C}^{i,t}, \{\mathsf{lab}_k^{i,t}\}_{k \in [\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left(\mathsf{ots}^{i,t,\alpha^*,\beta^*}, \{\mathsf{lab}_k^{i,t+1}\}_{k \in [\ell] \setminus \{h\}}\right)\right).$$
  - Sim sends $\{\widetilde{C}^{i,t}\}_{t \in [T]}$, and for each $j \in [n]$ and $k \notin [(j-1)\ell/n + 1, (j-1)\ell/n + (n-1)\kappa]$, it sends $\mathsf{lab}_k^{i,1}$. In addition, it sends for each $j, j' \in [n]$ such that $j \neq j'$ and $k \in [\kappa]$, $\left(\mathsf{Enc}(sk_{k,0}^{i,j,j'}, \mathsf{lab}_{\mathsf{GetIndex}(j,j',k)}^{i,1}), \mathsf{Enc}(sk_{k,1}^{i,j,j'}, \mathsf{lab}_{\mathsf{GetIndex}(j,j',k)}^{i,1})\right)$.

Output Computation. For every $i \in [n] \setminus H$, Sim obtains the second round message from $\mathcal{A}$ on behalf of the malicious parties. Subsequent to obtaining these messages, Sim uses the honest output computing procedure to see if the execution of garbled circuits proceeds consistently with the expected faithful execution. If the computation succeeds then, Sim sends $(\mathsf{generateOutput}, \mathsf{sid})$ to the ideal functionality. Otherwise, it sends $(\mathsf{abort}, \mathsf{sid})$.

Proof of Indistinguishability. We now show that the real execution and the simulated execution are computationally indistinguishable via a hybrid argument.

- $\underline{\mathsf{Hybrid}_0}$: This corresponds to the view of the adversary and the output of the honest parties in the real execution of the protocol.
- $\underline{\mathsf{Hybrid}_1}$: In this hybrid, we make the following changes:
  - **CRS Generation.** For each $i \in [n]$ such that $P_i \in C$, we sample $(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}^1_S(1^\lambda)$. We use the sampled crs^i to generate the crs.
  - **Input Extraction.** For each $i \in [n]$ such that $P_i \in C$ and for each $t \in A_i$, $\alpha, \beta \in \{0,1\}$, we compute $b^{i,t,\alpha,\beta} = \mathsf{Sim}^2_S(\mathsf{crs}^i, \mathsf{td}^i, \mathsf{otr}^{i,t,\alpha,\beta})$.
  - **Round-3 message from honest P_i where $i \in [n]$.** For each $t \in [T]$,
    * Let $\phi_t = (i^*, f, g, h)$.
    * For each $i \notin [n] \setminus \{i^*\}$, compute for each $\alpha, \beta \in \{0,1\}$, $\mathsf{ots}^{i^*,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{crs}^{i^*}, \mathsf{otr}^{i^*,t,\alpha,\beta}, \mathsf{lab}^{i,t}_{b^{i,t,\alpha,\beta}}, \mathsf{lab}^{i,t}_{b^{i,t,\alpha,\beta}})$. Here, if $P_{i^*} \in C$, then $b^{i^*,t,\alpha,\beta}$ is the extracted value. Otherwise, if $P_{i^*} \in H$, then $b^{i^*,t,\alpha,\beta} = v_{i^*}[h] \oplus \mathsf{NAND}(v_{i^*}[f] \oplus \alpha, v_{i^*}[g] \oplus \beta)$

  The indistinguishability between Hybrid_0 and Hybrid_1 is argued using the sender security of two-round oblivious transfer similar to Claim 5.15.
- $\underline{\mathsf{Hybrid}_2}$: In this hybrid, we make the following changes:
  - **CRS Generation.** For each $i \in [n]$, if $P_i \in H$, sample $(\mathsf{crs}^i, \mathsf{td}^i) \leftarrow \mathsf{Sim}^1_R(1^\lambda)$ and use crs^i to generate the crs.
  - **Round-2 message:** For each $i \in [n]$, if $P_i \in H$, and for each $t \in A_i$ and $\alpha, \beta \in \{0,1\}$, generate $(\mathsf{otr}^{i,t,\alpha,\beta}, \mu_0^{i,t,\alpha,\beta}, \mu_1^{i,t,\alpha,\beta}) \leftarrow \mathsf{Sim}^2_R(\mathsf{crs}^i, \mathsf{td}^i)$.

  The indistinguishability between Hybrid_1 and Hybrid_2 is argued using the equivocal receiver security of the oblivious transfer similar to Claim 5.16.
- $\underline{\mathsf{Hybrid}_3}$: In this hybrid, we do the following changes:
  - We initialize $\mathsf{aux} = \perp$.
  - We set $\mathsf{st}^* = \Big((0^{(n-1)\kappa} \| z_1^{\mathsf{part}}) || \ldots || (0^{(n-1)\kappa} \| z_n^{\mathsf{part}})\Big)$.
  - For every $i \in [n]$ such that $P_i \in C$:
    * For every $j \neq i$ such that $P_j \in H$ and for each $k \in [\kappa]$,
      · We intercept the message $(\mathsf{input}, (i,j,k), P_i, (\alpha_k^{i,j}, r_k^{i,j}))$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{dSel}}$.

- · We choose a random bit $y^{i,j}_{k,\alpha^{i,j}_k}$.
- · We set $\mathsf{st}^*[\mathsf{GetIndex}(i,j,k)] := y^{i,j}_{k,\alpha^{i,j}_k} \oplus r^{i,j}_k$.
- · We add $((i,j,k),(\alpha^{i,j}_k, y^{i,j}_{k,\alpha^{i,j}_k}))$ to aux.

∗ For every $j \neq i$ such that $P_j \in C$ and for each $k \in [\kappa]$,

- · We intercept the message $(\mathsf{Corrupt}, (i,j,k), \beta^{i,j}_k)$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{dSel}}$.
- · We set $\mathsf{st}^*[\mathsf{GetIndex}(i,j,k)] := \beta^{i,j}_k$.

– For each $i \in [n]$ such that $P_i \in H$,

- ∗ For each $j \in [n]$ such that $P_j \in C$ and for each $k \in [\kappa]$, we intercept the message $(\mathsf{input}, (i,j,k), P_j, (y^{i,j}_{k,0}, y^{i,j}_{k,1}))$ that $\mathcal{A}$ sends on behalf of corrupt P_j.
- ∗ We set $\mathsf{st}^*[\mathsf{GetIndex}(i,j,k)] := y^{i,j}_{k,\alpha^{i,j}_k} \oplus r^{i,j}_k$ where $\alpha^{i,j}_k, r^{i,j}_k$ and $(y^{i,j}_{k,0}, y^{i,j}_{k,1})$ (if $P_j \in H$) are computed using honest parties' randomness.
- ∗ We add $((i,j,k),(y^{i,j}_{k,0}, y^{i,j}_{k,1}))$ to aux if $P_j \in C$.

We note that st^* that is computed as above is consistent with the adversarial and the honest parties input/randomness in the correlations phase. Furthermore, the value aux contains the inputs and the outputs of adversarial parties during OT invocations with an honest party in the correlations phase. Also, we haven't made any changes to the distribution of the messages in Hybrid_3 (when compared to Hybrid_2) and hence, these two hybrids are identical.

- $\underline{\mathsf{Hybrid}_4 :}$ In this hybrid, we make the following changes:
  - For each $i,j \in [n]$ s.t. $i \neq j$ and for each $k \in [\kappa]$, we do the following:
    - ∗ Let $\eta := \mathsf{GetIndex}(i,j,k)$.
    - ∗ For every $s \in [n]$ such that $P_s \in H$, we sample $sk^{s,i,j}_{k,0}, sk^{s,i,j}_{k,1}$ using $\mathsf{Gen}(1^\lambda)$.
    - ∗ For every $s \in [n]$ such that $P_s \in C$, we intercept the message $(\mathsf{input}, (i,j,k), P_q, (sk^{s,i,j}_{k,0}, sk^{s,i,j}_{k,1}))$ that $\mathcal{A}$ sends on behalf of corrupt P_s.
    - ∗ We deliver $(\mathsf{output}, (i,j,k), (\mathsf{st}^*[\eta], \{sk^{s,i,j}_{k,\mathsf{st}^*[\eta]}\}_{s\in[n]})$ as the output from $\mathcal{F}_{\mathsf{dSel}}$ to $\mathcal{A}$.
  - For each $i \in [n]$ such that $P_i \in H$, for each $j,j' \in [n]$ such that $j \neq j'$ and $k \in [\kappa]$,
    - ∗ Let $\eta = \mathsf{GetIndex}(j,j',k)$.
    - ∗ We send $\left(\mathsf{Enc}(sk^{i,j,j'}_{k,0}, \mathsf{lab}^{i,1}_{\eta,\mathsf{st}^*[\eta]}), \mathsf{Enc}(sk^{i,j,j'}_{k,1}, \mathsf{lab}^{i,1}_{\eta,\mathsf{st}^*[\eta]})\right)$ in the final round.

  The indistinguishability between Hybrid_3 and Hybrid_4 is argued using the semantic security of the symmetric key encryption similar to Claim 5.19.

- $\underline{\mathsf{Hybrid}_{4+t}}$ for $t \in [0,T]$. This distribution is the same as hybrid Hybrid_{4+t-1} except we change the distribution of the garbled circuits (in the third round) that play a role in the execution of the t^{th} round of the protocol Φ; namely, the action $\phi_t = (i^*, f, g, h)$. We describe the changes more formally below.
  - In this hybrid, we complete the execution of Φ using honest party inputs and randomness. The messages on behalf of corrupted parties are generated via faithful execution. Specifically, we use st^* and start the mental execution of Φ. In this computation

phase, we generate the honest party messages using the inputs and random coins of the honest parties and generate the messages of the each malicious party P_i by executing $\mathsf{Faithful}\left(i, \mathsf{st}^*, \{b^{i,t,\alpha,\beta}\}_{t \in A_i, \alpha,\beta}\right)$.

– Let $\mathsf{Z} \in \{0,1\}^T$ be the transcript obtained using the above step. Let st^*_T be the public state of one of the corrupted parties at the end of faithful execution and let st^*_t be the public state of the parties at the end of the t-th round of the computation phase. Finally, let $\alpha^* := \mathsf{st}^*_T[f]$, $\beta^* := \mathsf{st}^*_T[g]$ and $\gamma^* := \mathsf{st}^*_T[h]$. In Hybrid_{4+t} we make the following changes with respect to hybrid Hybrid_{4+t-1}:

  * We make the following two changes in how we generate messages for other honest parties P_i (i.e., $P_i \in H \setminus \{P_{i^*}\}$). We do not generate four $\mathsf{ots}^{i^*,t,\alpha,\beta}$ values but just one of them; namely, we generate $\mathsf{ots}^{i^*,t,\alpha^*,\beta^*}$ as $\mathsf{OT}_2(\mathsf{crs}^{i^*}, \mathsf{otr}^{i^*,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t})$. Second, we generate the garbled circuit

$$\left(\widetilde{C}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left(\mathsf{ots}^{i^*,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]\setminus\{h\}}\right)\right),$$

  where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{C}^{i,t+1}$ (for any $t \leq T-1$) and for $t = T$, $\{\mathsf{lab}^{i,T+1}_{k,\mathsf{st}^*_T[k]} := \bot\}_{k\in[\ell]}$.

  * If $P_{i^*} \in C$ then skip these changes. We make two changes in how we generate messages on behalf of P_{i^*}. First, for all $\alpha, \beta \in \{0,1\}$, we set $\mu^{i^*,t,\alpha^*,\beta^*}$ as $\mu^{i^*,t,\alpha^*,\beta^*}_{\mathsf{Z}_t}$ rather than $\mu^{i^*,t,\alpha^*,\beta^*}_{v_{i^*}[h]\oplus\mathsf{NAND}(v_{i^*}[f]\oplus\alpha^*, v_{i^*}[g]\oplus\beta^*)}$ (note that these two values are the same when using the honest party's input and randomness). Second, it generates the garbled circuit

$$\left(\widetilde{C}^{i^*,t}, \{\mathsf{lab}^{i^*,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left((\alpha^*,\beta^*,\gamma^*), \mu^{i^*,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i^*,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]}\right)\right),$$

  where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{C}^{i,t+1}$ (for any $t \leq T-1$) and for $t = T$, $\{\mathsf{lab}^{i,T+1}_{k,\mathsf{st}^*_T[k]} := \bot\}_{k\in[\ell]}$.

The indistinguishability between Hybrid_{4+t-1} and Hybrid_{4+t} for every $t \in [T]$ is argued using the security of the garbled circuits similar to Claim 5.20.

- $\underline{\mathsf{Hybrid}_{5+T}}$: In this hybrid, we modify the output phase of the computation to execute the garbled circuits provided by $\mathcal{A}$ on behalf of the corrupted parties and see if the execution of garbled circuits proceeds consistently with the transcript Z. If the computation succeeds then for each $P_i \in H$, we instruct the parties in H to output the result of the output computation; else, we instruct them to output $\bot$. This hybrid is computationally indistinguishable to the previous hybrid from the authenticity of input labels property of garbled circuits.

- $\underline{\mathsf{Hybrid}_{6+T}}$: In this hybrid, we just change how the transcript Z, $\{z_i\}_{i\in H\}}$, and the value st^*_T are generated. Instead of generating these using honest party inputs in execution with a faithful execution of Φ, we generate it via the simulator Sim_Φ (of the maliciously secure protocol Φ) with aux as additional input. Specifically, we generate z_i as $(r_i \| 0^{\ell/n-m})$ where r_i is uniformly chosen random string of length m for each $P_i \in H$. We compute st^*_T as described in the simulation. To generate the transcript, we execute the simulator Sim_Φ on input $(H, \mathsf{st}^*, \mathsf{aux})$ where messages on behalf of each corrupted party P_i are generated using $\mathsf{Faithful}(i, \mathsf{st}^*, \{b^{i,t,\alpha,\beta}\}_{t \in A_i, \alpha, \beta})$. (Note that Sim_Φ might rewind Faithful. This can be achieved since Faithful is just a polynomial time interactive procedure that can also be rewound.). Note that the value aux contains the inputs and the outputs of the adversary in every OT invocation with an honest party in the correlations phase. It now follows from the statistical security of Φ that Hybrid_{6+T} is identically statistically close to Hybrid_{5+T}.

We note that Hybrid_{6+T} is identically distributed to $\mathrm{EXEC}_{\mathcal{F}_f, \mathsf{Sim}, \mathcal{Z}}$.

□

Acknowledgements. We thank Benny Applebaum and Yuval Ishai for useful discussions.

References

[ABG+20] Benny Applebaum, Zvika Brakerski, Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Separating two-round secure computation from oblivious transfer. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 71:1–71:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[ACJ17] Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 468–499. Springer, Heidelberg, August 2017.

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

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 119–135. Springer, Heidelberg, May 2001.

[App17] Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 1–44. Springer International Publishing, 2017.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 361–377. Springer, Heidelberg, August 2005.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus

Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193. Springer, Heidelberg, May 2017.

[BGI+18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In *ITCS 2018*, pages 21:1–21:21, January 2018.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, October 2012.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 500–532, 2018.

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

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 190–213. Springer, Heidelberg, August 2016.

[Can00a] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.

[Can00b] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. http://eprint.iacr.org/2000/067.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219, 2004.

[CCM98] Christian Cachin, Claude Crépeau, and Julien Marcil. Oblivious transfer with a memory-bounded receiver. In *39th FOCS*, pages 493–502. IEEE Computer Society Press, November 1998.

[CDFR17] Ignacio Cascudo, Ivan Damgård, Oriol Farràs, and Samuel Ranellucci. Resource-efficient OT combiners with active security. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 461–486. Springer, Heidelberg, November 2017.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *51st FOCS*, pages 541–550. IEEE Computer Society Press, October 2010.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, Heidelberg, August 2003.

[DHRS04] Yan Zong Ding, Danny Harnik, Alon Rosen, and Ronen Shaltiel. Constant-round oblivious transfer in the bounded storage model. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 446–472. Springer, Heidelberg, February 2004.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 74–94. Springer, Heidelberg, February 2014.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013.

[GIS18] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-round MPC: Information-theoretic and black-box. In *TCC 2018, Part I*, LNCS, pages 123–151. Springer, Heidelberg, March 2018.

[GLS15] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 63–82. Springer, Heidelberg, August 2015.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

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

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

[GS17] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In *58th FOCS*, pages 588–599. IEEE Computer Society Press, 2017.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. LNCS, pages 468–499. Springer, Heidelberg, 2018.

[HK12] Shai Halevi and Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. *Journal of Cryptology*, 25(1):158–193, January 2012.

[HKN+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 96–113. Springer, Heidelberg, May 2005.

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

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

[JKKR17] Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Ron Rothblum. Distinguisher-dependent simulation in two rounds and its applications. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 158–189. Springer, Heidelberg, August 2017.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988.

[Kus89] Eyal Kushilevitz. Privacy and communication complexity. In *30th FOCS*, pages 416–421. IEEE Computer Society Press, October / November 1989.

[LLW20] Huijia Lin, Tianren Liu, and Hoeteck Wee. Information-theoretic 2-round MPC without round collapsing: Adaptive security, and more. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 502–531. Springer, 2020.

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

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[PS16] Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 217–238. Springer, Heidelberg, October / November 2016.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In S. Jajodia and P. Samarati, editors, *ACM CCS 00*, pages 245–254. ACM Press, November 2000.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

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

A Universal Composition Framework

Below we briefly review the Universal Composition (UC) security. For full details see [Can01]. Most parts of this section has been taken verbatim from [CLP10]. A reader familiar with the notion of UC security can safely skip this section.

A.1 The basic model of execution

Following [GMR88, Gol01], a protocol is represented as an interactive Turing machine (ITM), which represents the program to be run within each participant. Specifically, an ITM has three tapes that can be written to by other ITMs: the input and subroutine output tapes model the inputs from and the outputs to other programs running within the same "entity" (say, the same physical computer), and the incoming communication tapes and outgoing communication tapes model messages received from and to be sent to the network. It also has an identity tape that cannot be written to by the ITM itself. The identity tape contains the program of the ITM (in some standard encoding) plus additional identifying information specified below. Adversarial entities are also modeled as ITMs.

We distinguish between ITMs (which represent static objects, or programs) and *instances of ITMs*, or ITIs, that represent interacting processes in a running system. Specifically, an ITI is an ITM along with an identifer that distinguishes it from other ITIs in the same system. The identifier consists of two parts: A session-identifier (SID) which identifies which protocol instance the ITM belongs to, and a party identifier (PID) that distinguishes among the parties in a protocol instance. Typically the PID is also used to associate ITIs with "parties", or clusters, that represent some administrative domains or physical computers.

The model of computation consists of a number of ITIs that can write on each other's tapes in certain ways (specified in the model). The pair (SID,PID) is a unique identifier of the ITI in the system.

We assume that all ITMs are probabilistic polynomial time (PPT). An ITM is PPT if there exists a constant $c > 0$ such that, at any point during its run, the overall number of steps taken by

M is at most n^c, where n is the overall number of bits written on the *input tape* of M in this run. (In fact, in order to guarantee that the overall protocol execution process is bounded by a polynomial, we define n as the total number of bits written to the input tape of M, *minus the overall number of bits written by M to input tapes of other ITMs.*; see [Can01].)

A.2 Security of protocols

Protocols that securely carry out a given task (or, protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in an adversarial environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an "ideal functionality," which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality. Below we overview the model of protocol execution (called the *real-life model*), the ideal process, and the notion of protocol emulation.

The model for protocol execution. The model of computation consists of the parties running an instance of a protocol Π, an adversary $\mathcal{A}$ that controls the communication among the parties, and an *environment* $\mathcal{Z}$ that controls the inputs to the parties and sees their outputs. We assume that all parties have a security parameter $n \in \mathbb{N}$. (We remark that this is done merely for convenience and is not essential for the model to make sense). The execution consists of a sequence of *activations*, where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some other ITM) is activated, and may write on a tape of at most *one* other participant, subject to the rules below. Once the activation of a participant is complete (i.e., once it enters a special waiting state), the participant whose tape was written on is activated next. (If no such party exists then the environment is activated next.)

The environment is given an external input z and is the first to be activated. In its first activation, the environment invokes the adversary $\mathcal{A}$, providing it with some arbitrary input. In the context of UC security, the environment can from now on invoke (namely, provide input to) only ITMs that consist of a single instance of protocol Π. That is, all the ITMs invoked by the environment must have the same SID and the code of Π.

Once the adversary is activated, it may read its own tapes and the outgoing communication tapes of all parties. It may either deliver a message to some party by writing this message on the party's incoming communication tape or report information to $\mathcal{Z}$ by writing this information on the subroutine output tape of $\mathcal{Z}$. For simplicity of exposition, in the rest of this paper we assume authenticated communication; that is, the adversary may deliver only messages that were actually sent. (This is however not essential as shown in [Can04, BCL$^+$05].)

Once a protocol party (i.e., an ITI running Π) is activated, either due to an input given by the environment or due to a message delivered by the adversary, it follows its code and possibly writes a local output on the subroutine output tape of the environment, or an outgoing message on the adversary's incoming communication tape.

In this work, we consider the setting of static corruptions. In the static corruption setting, the set of corrupted parties is determined at the start of the protocol execution and does not change during the execution.

The protocol execution ends when the environment halts. The output of the protocol execution is the output of the environment. Without loss of generality we assume that this output consists of only a single bit.

Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n,z,r)$ denote the output of the environment $\mathcal{Z}$ when interacting with parties running protocol Π on security parameter n, input z and random input $r = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1, r_2, \ldots$ as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$; $r_{\mathcal{A}}$ for $\mathcal{A}$, r_i for party P_i). Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n,z)$ random variable describing $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n,z,r)$ where r is uniformly chosen. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^*}$.

Ideal functionalities and ideal protocols. Security of protocols is defined via comparing the protocol execution to an *ideal protocol* for carrying out the task at hand. A key ingredient in the ideal protocol is the *ideal functionality* that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM (representing a "trusted party") that interacts with the parties and the adversary. More specifically, in the ideal protocol for functionality $\mathcal{F}$ all parties simply hand their inputs to an ITI running $\mathcal{F}$. (We will simply call this ITI $\mathcal{F}$. The SID of $\mathcal{F}$ is the same as the SID of the ITIs running the ideal protocol. (the PID of $\mathcal{F}$ is null.)) In addition, $\mathcal{F}$ can interact with the adversary according to its code. Whenever $\mathcal{F}$ outputs a value to a party, the party immediately copies this value to its own output tape. We call the parties in the ideal protocol dummy parties. Let $\Pi(\mathcal{F})$ denote the ideal protocol for functionality $\mathcal{F}$.

Securely realizing an ideal functionality. We say that a protocol Π *emulates* protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running Π, or it is interacting with S and parties running ϕ. This means that, from the point of view of the environment, running protocol Π is 'just as good' as interacting with ϕ. We say that Π *securely realizes* an ideal functionality $\mathcal{F}$ if it emulates the ideal protocol $\Pi(\mathcal{F})$. More precise definitions follow. A distribution ensemble is called *binary* if it consists of distributions over $\{0,1\}$.

Definition A.1. *Let* Π *and* ϕ *be protocols. We say that* Π *UC-emulates* ϕ *if for any adversary* $\mathcal{A}$ *there exists an adversary* $\mathcal{S}$ *such that for any environment* $\mathcal{Z}$ *that obeys the rules of interaction for UC security we have* $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \stackrel{c}{\approx} \text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$.

Definition A.2. *Let* $\mathcal{F}$ *be an ideal functionality and let* Π *be a protocol. We say that* Π *UC-realizes* $\mathcal{F}$ *if* Π *UC-emulates the ideal process* $\Pi(\mathcal{F})$.

A.3 Hybrid protocols

Hybrid protocols are protocols where, in addition to communicating as usual as in the standard model of execution, the parties also have access to (multiple copies of) an ideal functionality. Hybrid protocols represent protocols that use idealizations of underlying primitives, or alternatively make *trust assumptions* on the underlying network. They are also instrumental in stating the universal composition theorem. Specifically, in an $\mathcal{F}$-hybrid protocol (i.e., in a hybrid protocol with access to an ideal functionality $\mathcal{F}$), the parties may give inputs to and receive outputs from an unbounded number of copies of $\mathcal{F}$.

The communication between the parties and each one of the copies of $\mathcal{F}$ mimics the ideal process. That is, giving input to a copy of $\mathcal{F}$ is done by writing the input value on the input tape of that copy. Similarly, each copy of $\mathcal{F}$ writes the output values to the subroutine output tape of the corresponding party. It is stressed that the adversary does not see the interaction between the copies of $\mathcal{F}$ and the honest parties.

The copies of $\mathcal{F}$ are differentiated using their sub-session IDs (see UC with joint state [CR03]). All inputs to each copy and all outputs from each copy carry the corresponding sub-session ID. The model does not specify how the sub-session IDs are generated, nor does it specify how parties "agree" on the sub-session ID of a certain protocol copy that is to be run by them. These tasks are left to the protocol. This convention seems to simplify formulating ideal functionalities, and designing protocols that securely realize them, by freeing the functionality from the need to choose the sub-session IDs and guarantee their uniqueness. In addition, it seems to reflect common practice of protocol design in existing networks.

The definition of a protocol securely realizing an ideal functionality is extended to hybrid protocols in the natural way.

The universal composition operation. We define the universal composition operation and state the universal composition theorem. Let ρ be an $\mathcal{F}$-hybrid protocol, and let Π be a protocol that securely realizes $\mathcal{F}$. The composed protocol ρ^{Π} is constructed by modifying the code of each ITM in ρ so that the first message sent to each copy of $\mathcal{F}$ is replaced with an invocation of a new copy of Π with fresh random input, with the same SID (different invocations of $\mathcal{F}$ are given different sub-session IDs), and with the contents of that message as input. Each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of Π, with the contents of that message given to Π as new input. Each output value generated by a copy of Π is treated as a message received from the corresponding copy of $\mathcal{F}$. The copy of Π will start sending and receiving messages as specified in its code. Notice that if Π is a $\mathcal{G}$-hybrid protocol (i.e., ρ uses ideal evaluation calls to some functionality $\mathcal{G}$) then so is ρ^{Π}.

The universal composition theorem. Let $\mathcal{F}$ be an ideal functionality. In its general form, the composition theorem basically says that if Π is a protocol that UC-realizes $\mathcal{F}$ then, for any $\mathcal{F}$-hybrid protocol ρ, we have that an execution of the composed protocol ρ^{Π} "emulates" an execution of protocol ρ. That is, for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and protocol ρ^{Π} or with $\mathcal{S}$ and protocol ρ, in a UC interaction. As a corollary, we get that if protocol ρ UC-realizes $\mathcal{F}$, then so does protocol ρ^{Π}. [7]

Theorem A.3 (Universal Composition [Can01].). *Let $\mathcal{F}$ be an ideal functionality. Let ρ be a $\mathcal{F}$-hybrid protocol, and let Π be a protocol that UC-realizes $\mathcal{F}$. Then protocol ρ^{Π} UC-emulates ρ.*

An immediate corollary of this theorem is that if the protocol ρ UC-realizes some functionality $\mathcal{G}$, then so does ρ^{Π}.

A.4 The Common Reference/Random String Functionality

In the common reference string (CRS) model [CF01, CLOS02], all parties in the system obtain from a trusted party a reference string, which is sampled according to a pre-specified distribution D. The reference string is referred to as the *CRS*. In the UC framework, this is modeled by an ideal functionality $\mathcal{F}_{\mathsf{crs}}^{D}$ that samples a string ρ from a pre-specified distribution D and sets ρ as the CRS. $\mathcal{F}_{\mathsf{crs}}^{D}$ is described in Figure 11.

[7] The universal composition theorem in [Can01] applies only to "subroutine respecting protocols", namely protocols that do not share subroutines with any other protocol in the system.

Functionality $\mathcal{F}_{\mathsf{crs}}^{\mathbf{D}}$

$\mathcal{F}_{\mathsf{crs}}^{\mathbf{D}}$ runs with parties $P_1, \ldots P_n$ and is parameterized by a sampling algorithm D.

1. Upon activation with session id sid proceed as follows. Sample $\rho = D(r)$, where r denotes uniform random coins, and send $(\mathtt{crs}, sid, \rho)$ to the adversary.

2. On receiving $(\mathtt{crs}, sid)$ from the adversary, send $(\mathtt{crs}, sid, \rho)$ to every party.

Figure 11: The Common Reference String Functionality.

When the distribution D in $\mathcal{F}_{\mathsf{crs}}^{D}$ is sent to be the uniform distribution (on a string of appropriate length) then we obtain the common random string functionality denoted as $\mathcal{F}_{\mathsf{crs}}$.

A.5 General Functionality

We consider the general-UC functionality $\mathcal{F}$, which securely evaluates any polynomial-time (possibly randomize) function $f : (\{0,1\}^{\ell_{in}})^n \to (\{0,1\}^{\ell_{out}})^n$. The functionality $\mathcal{F}_f$ is parameterized with a function f and is described in Figure 12. In this paper we will only be concerned with the *static* corruption model.

Functionality $\mathcal{F}_{\mathbf{f}}$

$\mathcal{F}_f$ parameterized by an (possibly randomized) n-ary function f, running with parties $\mathcal{P} = \{P_1, \ldots P_n\}$ (of which some may be corrupted) and an adversary $\mathcal{S}$, proceeds as follows:

1. Each party P_i (and $\mathcal{S}$ on behalf of P_i if P_i is corrupted) sends $(\mathsf{input}, \mathsf{sid}, \mathcal{P}, P_i, x_i)$ to the functionality.

2. Upon receiving the inputs from all parties, evaluate $(y_1, \ldots y_n) \leftarrow f(x_1, \ldots, x_n)$. For every P_i that is corrupted send adversary $\mathcal{S}$ the message $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, y_i)$.

3. On receiving $(\mathsf{generateOutput}, \mathsf{sid}, \mathcal{P})$ from $\mathcal{S}$ the ideal functionality, outputs $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, y_i)$ to every P_i. (And ignores the message if inputs from all parties in $\mathcal{P}$ have not been received.)

Figure 12: General Functionality.

A.6 General Functionality with Input Dependent Abort

We also consider a weaker notion of security, called as *input-dependent abort*, where we allow an adversary to correlate the abort of an honest party with the inputs of all other honest parties. We follow the modeling used by [IKO+11] and describe the ideal functionality $\mathcal{F}_f^{\dagger}$ in Figure 13.

Functionality $\mathcal{F}_{\mathbf{f}}^{\dagger}$

$\mathcal{F}_f^{\dagger}$ parameterized by an (possibly randomized) n-ary function f, running with parties $\mathcal{P} = \{P_1, \ldots P_n\}$ (of which some may be corrupted) and an adversary $\mathcal{S}$, proceeds as follows:

1. Each party P_i (and $\mathcal{S}$ on behalf of P_i if P_i is corrupted) sends $(\mathsf{input}, \mathsf{sid}, \mathcal{P}, P_i, x_i)$ to the functionality.

2. $\mathcal{S}$ additionally sends $(\mathsf{predicate}, \mathsf{sid}, \phi)$ where ϕ denotes the description of a predicate $\phi : (\{0,1\}^{\ell_{in}})^n \rightarrow \{0,1\}$.

3. Upon receiving the inputs from all parties, evaluate $(y_1, \ldots y_n) \leftarrow f(x_1, \ldots, x_n)$ and z. For every P_i that is corrupted send adversary $\mathcal{S}$ the message $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, y_i)$.

4. On receiving $(\mathsf{generateOutput}, \mathsf{sid}, \mathcal{P})$ from $\mathcal{S}$, the ideal functionality computes $z = \phi(x_1, \ldots, x_n)$. If $z = 0$, it outputs $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, y_i)$ to every P_i. Else, if $z = 1$, it sends $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, \mathsf{abort})$ to every P_i (And ignores the message if inputs from all parties in $\mathcal{P}$ have not been received.)

Figure 13: General Functionality with Input Dependent Abort.