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

Arpita Patra[1] and Akshayaram Srinivasan[2]

[1] Indian Institute of Science
[2] 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 (OT). For the case of semi-honest adversaries, we make use of a two-round, semi-honest secure OT 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 OT in the common random/reference string model that satisfies a (mild) variant of adaptive security for the receiver.

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 [36] and Goldreich, Micali, and Wigderson [20] 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 [5] 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. [14] gave a construction of such a protocol based on indistinguishability obfuscation [4, 15] and subsequent work of Gordon et al. [21] improved the assumption to a witness encryption scheme [16]. Later, Mukherjee and Wichs [31] (and the subsequent works [9, 33]) gave a protocol based on the Learning with Errors assumption [35], Garg and Srinivasan [18] gave a construction from Bilinear maps and Boyle et al. [7, 8] gave a construction from the Decisional Diffie-Hellman (DDH) assumption. Finally, the works of Benhamouda and Lin [6] and Garg and Srinivasan [19] 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 [6, 19] made non-black-box use of a two-round OT. On the other hand, a recent work of Applebaum et al. [3] 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 [2, 17, 30]. These works left open the following intriguing question (which was explicitly mentioned in [3]):

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 *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 [34]. The main theorem we show for malicious adversaries is the following:

Informal Theorem 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 [19] 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 our MPC protocols in the semi-honest and the malicious setting.

Starting Point. Our work builds on the recent results of [6, 19] which gave constructions of a two-round MPC protocol from two-round OT. 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 OT. Unfortunately, these constructions [6, 19] 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 OT, 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 [29] rules out information-theoretic secure computation protocols for most functions in the dishonest majority setting. Furthermore, the work of Applebaum et al. [3] showed that such non-black-box use of OT 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 OT.

The work of Garg, Ishai, and Srinivasan [17] observed that if the parties apriori shared random OT correlations, then one can use the results of [28, 26] 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 [6, 19] gives rise to an MPC protocol that makes black-box use of cryptography. Garg et al. [17] also gave a method of generating such correlations in a single round using a primitive called *non-interactive OT*. 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 OT, and use the next two rounds to implement the round-collapsing compiler of [6, 19]. However, a non-interactive OT is a very strong primitive and it is not known whether this can be constructed generically from a two-round OT.

Double Selection Functionality. If we abstract out the other details from [17], 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. [17] can be viewed as giving a three-round protocol for the double selection functionality based on non-interactive OT. The goal of this work is to give such a protocol based only on black-box use of a two-round OT.

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 [2] 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* [3]. 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 OT.

Key Idea: "Cascading OT." 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 OT. Indeed, this implies that we need a mechanism to compute the output of a degree-3 function in two rounds using a two-round OT 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 OT" 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, each 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 OT" 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 OT.

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 [17]. Alternatively, we can use the above idea of cascading OT 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 [8, 17, 3] who showed a round-preserving black-box reduction from a semi-honest protocol for computing general functions to a secure protocol for $\mathsf{3MULTPlus}$ functionality. In the main body, we construct a protocol for securely computing $\mathsf{3MULTPlus}$ 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 OT 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. [25].

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 $\{\mathsf{ots}^i_0, \mathsf{ots}^i_1, \mathsf{ots}^i, \mathsf{ots}'^i\}$ 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 $(y^i_0, y^i_1, \mathsf{mask}^i)$ 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 a trick from [27], wherein P_1, in the first round, uses a t-out-of-m OT 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.[3]

Need for Equivocal Receiver Security. Here, another technical issue arises and to solve this, we need the OT 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 OT that is secure against adversaries that can adaptively corrupt the receiver, or it can be obtained from a dual-mode public-key encryption scheme [34].

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

[3] Here, we need to additionally ensure that malicious P_3 is generating the shares correctly. Hence, we make use of a pairwise verifiable secret sharing based on bivariate polynomials and do additional checks on the shares to ensure that the sharing is done correctly.

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. The 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 both P_1 and P_2 to learn $(x_\alpha \oplus r)$.

A statistically secure protocol for the above functionality is obtained by first implementing the information-theoretic OT combiner protocol from [12] to extract "pure" OT correlations from the above "leaky" OT correlations and then use the information-theoretic two-party protocols [28, 26, 24] 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 [6, 19] 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 [19]: 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 [17] 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 [28, 26] 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 OT with equivocal receiver security.

3 Preliminaries

We recall some standard cryptographic definitions in this section. Let λ denote the security parameter. We give the standard definition for negligible functions, computational indistinguishability and the UC framework [11] in the full version.

3.1 Oblivious Transfer

In this paper, we consider a 1-out-of-2 OT, similar to [10, 32, 1, 13, 34, 22] 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 OT. 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, $\{\mathsf{otr} : \omega \leftarrow \{0,1\}^*, \mathsf{otr} := \mathsf{OT}_1(1^\lambda, 0; \omega)\} \overset{c}{\approx} \{\mathsf{otr} : \omega \leftarrow \{0,1\}^*, \mathsf{otr} := \mathsf{OT}_1(1^\lambda, 1; \omega)\}$.
- **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, $\{\beta, \omega \leftarrow \{0,1\}^*, \mathsf{OT}_2(1^\lambda, \mathsf{otr}, K_0, K_1)\} \overset{c}{\approx} \{\beta, \omega \leftarrow \{0,1\}^*, \mathsf{OT}_2(1^\lambda, \mathsf{otr}, L_0, L_1)\}$ where $\mathsf{otr} := \mathsf{OT}_1(1^\lambda, \beta; \omega)$.

Remark 1. 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 OT 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 [19]. 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 OT 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 $\beta \in \{0,1\}$ 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:

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

- **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 $K^i_0, K^i_1 \in \{0,1\}^\lambda$ for $i \in [n]$ where $n = \mathsf{poly}(\lambda)$, PPT adversary $\mathcal{A}$ and any PPT distinguisher D, we have:

$$\Big| \Pr[\mathrm{Expt}_1 = 1] - \Pr[\mathrm{Expt}_2 = 1] \Big| \le \mathsf{negl}(\lambda).$$

```
Expt_1:
    $\mathsf{crs} \leftarrow K_{\mathsf{OT}}(1^\lambda)$
    $\{\mathsf{otr}^i\}_{i\in[n]} \leftarrow \mathcal{A}(\mathsf{crs})$
    $\big\{\mathsf{ots}^i \leftarrow \mathsf{OT}_2(\mathsf{crs}, \mathsf{otr}^i, (K^i_0, K^i_1))\big\}_{i\in[n]}$
    Output $D(\mathsf{crs}, \{\mathsf{ots}^i\}_{i\in[n]})$

Expt_2:
    $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_S^1(1^\lambda)$
    $\{\mathsf{otr}^i\}_{i\in[n]} \leftarrow \mathcal{A}(\mathsf{crs})$
    $\beta_i := \mathsf{Sim}_S^2(\mathsf{crs}, \mathsf{td}, \mathsf{otr}^i)\ \forall i \in [n]$
    $L^i_0 := K^i_{\beta_i}$ and $L^i_1 := K^i_{\beta_i}$
    $\big\{\mathsf{ots}^i \leftarrow \mathsf{OT}_2(\mathsf{crs}, \mathsf{otr}, (L^i_0, L^i_1))\big\}_{i\in[n]}$
    Output $D(\mathsf{crs}, \{\mathsf{ots}^i\}_{i\in[n]})$
```

Remark 2. We note that a two-round malicious secure OT with equivocal receiver security implies a standard two-round malicious OT that implements the ideal OT functionality.

We recall the definitions of garbled circuits, non-interactive secure computation and some properties of symmetric bivariate polynomial in Appendix **??**.

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 OT 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 extend it for the case of general functions by relying on the results from [8, 17, 3].

4.1 First Step: 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 3. *There is an efficient three-round protocol* $\Pi_{\mathsf{3MULTPlus}}$ *(Figure 1) that makes black-box use of a two-round, semi-honest OT 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 describe a three-round protocol for securely computing $\mathcal{F}_{\mathsf{3MULTPlus}}$ against semi-honest adversaries making black-box access to a 2-round semi-honest OT. We give an informal description below.

At a high-level, the degree-3 computation is achieved by cascading OT messages i.e., generating a sender OT message where the inputs are themselves two other sender OT messages. Since OT enables degree-2 computation, cascading OT enables us to compute the result of a degree-3 computation. The main novelty 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.

In the first round, P_1, acting as a receiver, publishes an OT receiver message otr that encodes its input x_1. In parallel, P_2, first splits x_2 into two additive shares $(x_{2,0}, x_{2,1})$ and then publishes two OT receiver messages, $\mathsf{otr}_0, \mathsf{otr}_1$ where otr_b encodes $x_{2,b}$. In the second round, P_3 splits its input x_3 into two additive shares, $x_{3,0}, x_{3,1}$. It then prepares two OT sender messages with respect to the receiver messages $\mathsf{otr}_0, \mathsf{otr}_1$ where the sender inputs used in both these messages are given by $(x_{3,0}, x_{3,1})$. Let these OT messages 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 the sender inputs 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 conclude that P_1 at the end of the second round can compute $x_{3,x_{2,x_1}}$ 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 $x_{2,0,0}, x_{2,0,1}$ which are an additive secret sharing 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 this can be shown to be an additive secret sharing of the output. In the final round, each party broadcasts this value and this enables 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 in the case where there exists more than one honest party in the set 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 $x_1 x_2 x_3 + y_1 + y_2 + y_3$.
Primitive: A two-round semi-honest secure OT protocol $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$.

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 $(\delta^i_1, \delta^i_2, \delta^i_3)$ and sends the share δ^i_j to party P_j for $j \in [3] \setminus \{i\}$ via private channels.[a]

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 $z_1 + z_2 + z_3$.

[a] We can simulate a single round of private channel messages in two rounds over public channels by making use of a two-round OT.

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

We show the correctness and security in Lemma 1-2.

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

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,

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

This completes the proof of correctness.

Lemma 2 (Security). *Protocol* $\Pi_{\mathsf{3MULTPlus}}$ *securely computes* $\mathcal{F}_{\mathsf{3MULTPlus}}$ *against a semi-honest adversary corrupting an arbitrary subset of parties.*

We defer the proof to the full version.

4.2 Second Step: Protocol for Arbitrary Functions

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

Theorem 4 ([8, 17, 3]). *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 3 and the UC composition theorem [11], we get the following.

Corollary 1. *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 OT 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 OT 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. 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 $3 \leq i \leq n$, it receives $(z_0^i, z_1^i) \in \{0,1\}^{\lambda} \times \{0,1\}^{\lambda}$ from P_i. $\mathsf{dSelPri}$ delivers $(y_\alpha, \{z^i_{y_\alpha \oplus r}\}_{3 \leq i \leq n})$ 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, which implies the subsequent corollary via the results from [26, 24]

Theorem 5. *There exists a two-round protocol $\Pi^\dagger_{\mathsf{dSelPri}}$ (Figure 4) that UC-realizes $\mathcal{F}^\dagger_{\mathsf{dSelPri}}$ in the $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$ (Figure 3) hybrid model making black-box access to a two-round, malicious-secure OT with equivocal receiver security.*

Corollary 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 OT 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 $s'_j = \perp$ 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 $\Pi^\dagger_{\mathsf{dSelPri}}$. 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, assuming P_3 as the lone provider of a pair z_0, z_1. This version, in fact, is identical to the first two rounds of the construction for "double-selection" functionality implementing "cascaded OT" described in Section 2.1.

Now, to make the 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 [25] 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_2 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 OT [26, 24], 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 $(y_\alpha, \{z^i_{y_\alpha \oplus r}\}_{3 \leq i \leq n})$.

Primitives: (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 3.1). 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 $(\overline{\mathsf{otr}}, \overline{\mu}) \leftarrow \mathsf{OT}_1(\mathsf{crs}^1, r)$. 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 $\mathcal{F}_{(m,p)\text{-}\mathsf{RaOT}}$.
- Let $J_i \subseteq [m]$ such that for each $j \in J_i$, $\overline{s}_j^i \neq \bot$.
- 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 $\mathsf{CheckValid}(\mathsf{crs}^i, \mathsf{otr}_\gamma^{i,j}, (f_\gamma^{i,j}(x), \mu_\gamma^{i,j}))$ (see Sec. 3.1 for $\mathsf{CheckValid}$) 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 proves Theorem 5. We defer the proof to the full version.

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

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 [19]. In this subsection, we recall this notion and present a slightly modified version given in [17]. Further, these two steps will build upon the round-collapsing compiler of [19].

Specification of a Conforming Protocol. Consider an n-party deterministic[4] MPC protocol Φ between parties $P_1, \ldots, P_n$ with inputs $x_1, \ldots, x_n$, respectively computing some function $f(x_1, \ldots, x_n)$. 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: pre-processing, computation and output.

- **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

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

secret information and broadcasts z_i to every other party. We require that $v_i[k] = 0$ for all $k \in [\ell] \backslash \{(i-1)\ell/n + 1, \dots, i\ell/n\}$.[5]

– **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 $t \neq t'$, 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 which 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 $\mathsf{post}(\mathsf{st})$.

We now recall the following theorem proved in [19, 17].

Theorem 6 ([19, 17]). *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*)*[6] *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).*

5.3 Second Step: Special Functionality with Standard Security

In this subsection, we define the n-party version of the double-selection functionality $\mathcal{F}_{\mathsf{dSel}}$ in Figure 5 and give a three-round protocol for securely realizing this functionality. The main theorem we prove in this subsection is given below.

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 $(z_0^i, z_1^i) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$ 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. $\mathcal{F}_{\mathsf{dSel}}$ proceeds as follows:

[5] Here, we slightly differ from the formulation used in [19, 17]. 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 $v_i[(i-1)\ell/n + 1, (i-1)\ell/n + m]$.

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

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 functionality delivers $(\mathsf{output}, \mathsf{sid}, P_i, \mathsf{out})$ to every honest 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 honest P_i. (And ignores the message if inputs from all parties in $\{P_1, \ldots, P_n\}$ have not been received.)

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

Theorem 7. *There exists a three-round protocol Π_{dSel} (Figure 7) 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}. The primary challenge in Π_{dSel}, over $\Pi^{\dagger}_{\mathsf{dSelPri}}$, is to keep any corrupt P_2's behaviour, as an OT sender, in check. We resort to an OT combiner protocol [23, 12], that guarantees generation of a secure OT correlation given a number of leaky OTs, as formalized by functionality $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$ in Figure 6.

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

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 $(\mathsf{sender}, \mathsf{sid}, P_2, (K, \{(s^i_0, s^i_1)\}_{i\in[\kappa]}))$ to the functionality where $K \subseteq [\kappa]$ is a set of size at most λ and $s^i_b \in \{0,1\}$ for each $i \in [\kappa]$ and $b \in \{0,1\}$.
- On receiving both these messages, the functionality computes $\mathsf{out}_1 := \{(\alpha_i, s^i_{\alpha_i})\}_{i\in[\kappa]}$ 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 $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$

In keeping with the goal of publishing a masked version of P_1's selected input of P_2, i.e. $y_\alpha + r$, we slightly stretch the goal of OT combiner from realizing a secure OT correlation to realizing a simple two-party functionality captured by $\mathcal{F}_{\mathsf{OTplus}}$. $\mathcal{F}_{\mathsf{OTplus}}$ 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 realizing $\mathcal{F}_{\mathsf{OTplus}}$ in the $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$-hybrid model is guaranteed from an OT combiner protocol followed by a secure computation protocol in the OT-hybrid model [28, 24].

Theorem 8 ([12, 28, 24]). *Let $\kappa = \Omega(\lambda)$ and consider the $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$ 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 $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$ functionality. Furthermore, the inputs to $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$ 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.*

While Theorem 8 guarantees a protocol for $\mathcal{F}_{\mathsf{OTplus}}$, it may be a multi-round protocol and it is not clear how Π_{dSel} can use this for its goal to realize $\mathcal{F}_{\mathsf{dSel}}$. Here, we invoke the round-collapsing compiler of [19, 17] (for an informal discussion, refer to Appendix **??**) on a conforming protocol obtained from the protocol implied by Theorem 8 in $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$-hybrid model. To be specific, Theorem 8 implies the following protocol for realizing $\mathcal{F}_{\mathsf{OTplus}}$:

- **Call to $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$ 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 $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$ functionality, P_1 and P_2 interact with each other using the *statistically-secure* protocol Π_{OTplus} (from Theorem 8) that realizes the $\mathcal{F}_{\mathsf{OTplus}}$ functionality. In this protocol, P_1's input is given by $((\alpha, r), (s^1_{\alpha_1}, \alpha_1), \ldots, (s^\kappa_{\alpha_\kappa}, \alpha_\kappa))$ 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 a result of the transformation given in Theorem 6 to the protocol Π_{OTplus} (as above). We assume w.l.o.g. that the input of P_1 in Φ is of the form $(s^i_{\alpha_1}, \ldots, s^i_{\alpha_\kappa}, \alpha_1, \ldots, \alpha_k, \alpha, r)$ 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 (if either party has not aborted and this information is public from st).

Now to enable Π_{dSel} to achieve the larger goal of publishing "doubly-selected" inputs of $P_3, \ldots, P_n$, all that is needed from $P_3, \ldots, P_n$ is to take part in Φ and listen to the conversation. That is, 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 $\ell/2$. Specifically, we introduce another layer of garbled circuits for the parties P_3 to P_n that takes st as input, has z_0^i, z_1^i hardwired and outputs $z^i_{\mathsf{st}[\ell/2]}$ if st does not indicate an abort of P_1 or P_2. W.l.o.g., we can assume that st contains this information on abort. To tackle a malicious behaviour of P_i, we make them commit to z_0^i, z_1^i via OT receiver messages in the first round and reveal the opening information via the garbled circuit.

There are two missing blocks now: (a) how to create the correlation of a $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$ functionality (since Φ runs given the output of $\mathcal{F}_{\kappa\text{-}\mathsf{LeakyOT}}$) and (b) how to release the labels corresponding to the initial public joint state for every party's garbled circuit in 3 rounds. Both are resolved through κ calls to $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ functionality (recall that κ is the OT combiner parameter). Π_{dSel} runs 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_0^k, s_1^k) and the inputs for the rest of parties being equal to a pair of secret keys for a SKE scheme (looking ahead, these keys will enable release of the first set of labels). These κ executions of $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ 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.

We now explain how to release the labels corresponding to the initial public joint state for every party's garbled circuit in 3 rounds. This is where we use the secret keys 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 $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ 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 first set of garbled circuits. This will trigger evaluation of the bunch of circuits emulating Φ.

Lastly, we consider the $\mathcal{F}^{\dagger}_{\mathsf{dSelPri}}$ functionality instantiated with $n+1$ parties with 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_0^2, z_1^2).

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 $3 \le i \le n$, P_i inputs $(z_0^i, z_1^i) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$.

Output: Every party outputs $(y_\alpha \oplus r, \{z^i_{y_\alpha \oplus r}\}_{3 \le i \le n})$.

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 3.1). 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 6 to Π_{OTplus} as discussed. (d) Garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$ (see Section **??**) (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,
- 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 $\mathsf{st} := 0^\kappa \| (z_1^{\mathsf{part}} \| z_2)$.
- 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 8) $(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \{\mu^{i,t,\alpha,\beta}\}_{\alpha,\beta}, \perp, \mathsf{lab}^{i,t+1}])$.
  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 $(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \perp, \{\mathsf{ots}^{i^*,t,\alpha,\beta}\}_{\alpha,\beta}, \mathsf{lab}^{i,t+1}])$.

- Each P_i sends $(\{\widetilde{C}^{i,t}\}_{t\in[T]}, \{\mathsf{lab}^{i,1}_{k,\mathsf{st}[k]}\}_{k\in[\kappa+1,\ell]})$ 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 $\mathsf{CheckValid}$ algorithm outputs 0 then abort. Otherwise, output $(\mathsf{st}[\ell/2], \{z^j_{\mathsf{st}[\ell/2]}\}_{j\in[3,n]})$.

[a] 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 $C^{i,t}$ and ChkC^i

Input of $C^{i,t}$: st

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

Code of $C^{i,t}$:
- 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 $(\mathsf{ots}^{i^*,t,\mathsf{st}[f],\mathsf{st}[g]}, \{\mathsf{lab}_{k,\mathsf{st}[k]}\}_{k\in[\ell]\setminus\{h\}})$.

Input of ChkC^i: st

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

- 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 8: Circuit $C^{i,t}$ and ChkC^i

Lemma 4. *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 $\mathcal{Z}$,* $\mathrm{EXEC}_{\mathcal{F}_{\mathsf{dSel}},\mathsf{Sim},\mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}_{\Pi_{\mathsf{dSel}},\mathcal{A},\mathcal{Z}}$

We defer the proof of this lemma to the full version.

5.4 Third Step: Bootstrapping from Special to General Functions

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 shown here is the following.

Theorem 9. *Let f be a n-party functionality. There exists a protocol Π_f (Figure 9) 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 $\mathcal{F}_{\mathsf{dSel}}$-hybrid model.*

Building Π_f. The protocol Π_f is obtained as a result of applying the round-collapsing compiler in [19, 17] to perfect/statistical protocols in the OT-correlations model (e.g., [28, 26]) which have 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 [28, 26] in the OT correlations model to securely compute f.

Let Φ be the conforming protocol obtained as a result of the transformation in Theorem 6 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.

Building on the round-collapsing compiler of [19, 17] (see Appendix **??** for an informal description of the compiler), the main challenge in Π_f is in making the first set of labels for the joint state available within 3 rounds. Unlike [19, 17], 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 3.1), (b) Functionality $\mathcal{F}_{\mathsf{dSel}}$ (c) The conforming protocol Φ obtained as a result of the transformation in Theorem 6 to Π as discussed (c) Garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$ (see Section **??**) (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 $(\mathsf{input}, (i, j, k), P_j, (y_{k,0}^{i,j}, y_{k,1}^{i,j})$ to $\mathcal{F}_{\mathsf{dSel}}$ where $y_{k,0}^{i,j}, y_{k,1}^{i,j}$ are uniformly chosen bits.
  - For every $s \in [n]$, P_s inputs $(\mathsf{input}, (i, j, k), P_s, (sk_{k,0}^{s,i,j}, sk_{k,1}^{s,i,j}))$ to $\mathcal{F}_{\mathsf{dSel}}$ where $sk_{k,0}^{s,i,j}, sk_{k,1}^{s,i,j}$ are sampled using $\mathsf{Gen}(1^\lambda)$.

Round-2: In the second round, every P_i does the following
- It sets $x_i^{\mathsf{part}} := (x_i, \{\alpha_k^{i,j}, y_{k,0}^{j,i}, y_{k,1}^{j,i}\}_{j\in[n]\setminus\{i\}, k\in[\kappa]})$.
- 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 $\mathsf{st} = \Big((0^{(n-1)\kappa} \| z_1^{\mathsf{part}}) \| \ldots \| (0^{(n-1)\kappa} \| z_n^{\mathsf{part}})\Big)$.
- It sets $\mathsf{lab}^{i,T+1} := \{\mathsf{lab}_{k,0}^{i,T+1}, \mathsf{lab}_{k,1}^{i,T+1}\}_{k\in[\ell]}$ where for each $k \in [\ell]$ and $b \in \{0,1\}$, $\mathsf{lab}_{k,b}^{i,T+1} := \bot$.
- **for** each t from T down to 1,
  1. Let ϕ_t as (i^*, f, g, h).
  2. If $i = i^*$, then it computes $(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \{\mu^{i,t,\alpha,\beta}\}_{\alpha,\beta}, \bot, \mathsf{lab}^{i,t+1}])$ (where $C^{i,t}$ is described in Figure 8).
  3. If $i \neq i^*$ then for every $\alpha, \beta \in \{0,1\}$, it sets $\mathsf{ots}^{i^*,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{crs}^{i^*}, \mathsf{otr}^{i^*,t,\alpha,\beta}, \mathsf{lab}_{h,0}^{i,t+1}, \mathsf{lab}_{h,1}^{i,t+1})$ and computes $(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \bot, \{\mathsf{ots}^{i,t,\alpha,\beta}\}_{\alpha,\beta}, \mathsf{lab}^{i,t+1}])$ (where $C^{i,t}$ is described in Figure 8).
- 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 $\mathsf{lab}_{k,\mathsf{st}[k]}^{i,1}$. In addition, P_i broadcasts for each $j, j' \in [n]$ such that $j \neq j'$ and $k \in [\kappa]$, $\Big(\mathsf{ct}_{k,0}^{i,j,j'} = \mathsf{Enc}(sk_{k,0}^{i,j,j'}, \mathsf{lab}_{\mathsf{GetIndex}(j,j',k),0}^{i,1}), \mathsf{ct}_{k,1}^{i,j,j'} = \mathsf{Enc}(sk_{k,1}^{i,j,j'}, \mathsf{lab}_{\mathsf{GetIndex}(j,j',k),1}^{i,1})\Big)$.

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 $(\mathsf{output}, (j,j',k), P_i, (z_\eta, \{sk_{k,z_\eta}^{s,j,j'}\}_{s\in[n]}))$ from $\mathcal{F}_{\mathsf{dSel}}$ functionality.
  2. Reset $\mathsf{st}[\eta] = z_\eta$.
  3. For each $s \in [n]$, set $\mathsf{lab}_{\eta,\mathsf{st}[\eta]}^{s,1} \leftarrow \mathsf{Dec}(sk_{k,\mathsf{st}[\eta]}^{s,j,j'}, \mathsf{ct}_{k,\mathsf{st}[\eta]}^{s,j,j'})$.
- For every $j \in [n]$, let $\widetilde{\mathsf{lab}}^{j,1} = \{\mathsf{lab}_{k,\mathsf{st}[k]}^{j,1}\}_{k\in[\ell]}$, where $\{\mathsf{lab}_{k,\mathsf{st}[k]}^{j,1}\}_{k\in[(j-1)\ell/n+1,(j-1)\ell/n+(n-1)\kappa]}$ 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 $(\mathsf{ots}, \{\mathsf{lab}_{k,\mathsf{st}[k]}^{j,t+1}\}_{k\in[\ell]\setminus\{h\}}) := \mathsf{Eval}(\widetilde{C}^{j,t}, \widetilde{\mathsf{lab}}^{j,t})$.
     (b) Recover $\mathsf{lab}_{h,\mathsf{st}[h]}^{j,t+1} := \mathsf{OT}_3(\mathsf{crs}^{i^*}, \mathsf{ots}, (\gamma, \mu))$.
     (c) Set $\widetilde{\mathsf{lab}}^{j,t+1} := \{\mathsf{lab}_{k,\mathsf{st}[k]}^{j,t+1}\}_{k\in[\ell]}$.
- Output $\mathsf{post}(\mathsf{st}, v_i)$.

Figure 9: Protocol Π_f

Lemma 5. *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 $\mathcal{Z}$,* $\mathrm{EXEC}_{\mathcal{F}_f,\mathsf{Sim},\mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}_{\Pi_f,\mathcal{A},\mathcal{Z}}$

We give the proof of this lemma in the full version.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
12. 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.
13. 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.

14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. LNCS, pages 468–499. Springer, Heidelberg, 2018.
20. 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.
21. 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.
22. Shai Halevi and Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. *Journal of Cryptology*, 25(1):158–193, January 2012.
23. 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.
24. 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.
25. 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.
26. 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.
27. 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.
28. Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988.
29. Eyal Kushilevitz. Privacy and communication complexity. In *30th FOCS*, pages 416–421. IEEE Computer Society Press, October / November 1989.

30. 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.
31. 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.
32. Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.
33. 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.
34. 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.
35. 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.
36. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.