Round-Optimal Black-Box Multiparty Computation from Polynomial-Time Assumptions

Michele Ciampi[1], Rafail Ostrovsky[2],
Luisa Siniscalchi[3], and Hendrik Waldner[4]

[1] The University of Edinburgh, Edinburgh, UK
mciampi@ed.ac.uk
[2] University of California, Los Angeles, US
rafail@cs.ucla.edu
[3] Technical University of Denmark, Kongens Lyngby, Denmark
luisi@dtu.dk
[4] University of Maryland, College Park, US
hwaldner@umd.edu

Abstract. A central direction of research in secure multiparty computation with dishonest majority has been to achieve three main goals:

1. reduce the total number of rounds of communication (to four, which is optimal);
2. use only polynomial-time hardness assumptions, and
3. rely solely on cryptographic assumptions in a black-box manner.

This is especially challenging when we do not allow a trusted setup assumption of any kind. While protocols achieving two out of three goals in this setting have been designed in recent literature, achieving all three *simultaneously* remained an elusive open question. Specifically, it was answered positively only for a restricted class of functionalities. In this paper, we completely resolve this long-standing open question. Specifically, we present a protocol for all polynomial-time computable functions that does not require any trusted setup assumptions and achieves all three of the above goals simultaneously.

Table of Contents

1 Introduction . . . 3
- 1.1 Our Contribution . . . 3
- 1.2 Technical Overview . . . 4

2 Preliminaries . . . 8
- 2.1 Oblivious Transfer . . . 8
- 2.2 MPC Definitions . . . 11
- 2.3 List MPC . . . 13
- 2.4 Garbled Circuits . . . 16
- 2.5 Symmetric Encryption and Message Authentication . . . 17
- 2.6 Outer Protocol . . . 18

3 Single Receiver, Multiple Senders Oblivious Transfer (1RnS OT) . . . 19
- 3.1 1RnS Oblivious Transfer from 1-out-of-2 Oblivious Transfer . . . 20
- 3.2 Definition of nRmS OT . . . 29
- 3.3 Formal Description of $\mathsf{OT}_{\mathsf{nRmS}}$ . . . 30

4 Inner Protocol . . . 31
- 4.1 Definition of the Inner-Protocol . . . 31
- 4.2 Realization of the Inner-Protocol . . . 34
- 4.3 Oblivious Transfer with Equivocal Receiver Security . . . 34
- 4.4 Protocol for the Double Selection Functionality DS [PS21] . . . 35
- 4.5 Conforming Protocols . . . 37
- 4.6 Formal Description of the Three Round Inner Protocol . . . 38

5 Our Four-Round MPC Protocol . . . 53

1 Introduction

Secure multiparty computation (MPC) [Yao86, GMW87] enables a group of parties to jointly evaluate any function over their inputs in a privacy preserving way. More precisely, even if a subset of the parties collude to attack the protocol no one learns anything beyond the output of the function. Since its introduction, there has been a long sequence of works that aims to design efficient protocols while relying on well-understood cryptographic assumptions and minimizing the need for trusted setups [GMW87, Kil88, IPS08, BMR90, KOS03, KO04, Pas04, PW10, Wee10, Goy11, GMPP16, ACJ17, BHP17, DGL+21, GLO+21, COWZ22]. In 2004, Katz and Ostrovsky [KO04] showed that, to securely realize the coin tossing functionality in the two-party case in the plain model with static corruption and black-box simulation,[5] the parties need to interact in at least five rounds (where in each round only one party speaks). Following this work, Garg et al. [GMPP16] established that if the parties are allowed to speak in the same round (*simultaneous communication model*) then at least four rounds of communication are needed to securely realize the coin tossing functionality. This result trivially extends to the n party case, where up to $(n-1)$ parties can be corrupted and where, in each round, all the parties are allowed to broadcast a message. Garg et al. also showed that four rounds are sufficient to realize any two-party functionality while making non-black-box use of the underlying cryptographic primitives. A sequence of works tried to match the same lower bound for the multiparty setting [ACJ17, BHP17], and finally [BGJ+18, HHPV18] showed the first four-round protocols based on polynomial time number theoretic assumptions. This result was later improved in [CCG+20] proposing a four-round scheme based on maliciously secure oblivious transfer (OT). However, all these results require non-black-box use of the underlying primitives.

It is natural to ask whether it is possible to achieve similar results while relying on the underlying primitives in a black-box way. This question has been investigated extensively in the past years [IPS08, Wee10, Goy11], but only recently Ishai, Khurana, Sahai, and Srinivasan [IKSS21] came close to matching the four-round lower bound, proposing a five-round protocol that makes black-box use of secure oblivious transfer and public-key encryption (PKE) with pseudorandom public keys. This result was later improved in [IKSS23] where the same authors construct a (black-box) four-round protocol, requiring sub-exponential security of the underlying assumptions. In a concurrent and independent work [COSW23a], propose a black-box protocol that requires polynomial-time assumptions but only realizes a restricted set of functionalities (i.e., input-less functionalities). This leaves open the following question:

> *Is it possible to construct a four-round protocol for securely evaluating generic multiparty functionalities while making black-box use of polynomial-time hardness assumptions?*

1.1 Our Contribution

We answer the above question positively, proposing a four-round MPC protocol that realizes generic functionalities in the plain model in the dishonest majority setting while making

[5] All the results presented in this paper are with respect to black-box simulation, dishonest majority, static corruption, and in the plain model. Hence, we will not specify this in the remainder of the paper.

black-box use of any perfect correct two-round OT protocol with statistical sender privacy. This notion of OT was introduced in [NP01, AIR01] and requires computational security for the receiver's message, i.e., it is computationally hard to distinguish whether the receiver's input bit b is equal to 0 or 1. Moreover, regardless of the receiver's first message, the sender's message hides (statistically) at least one of its inputs (denoted by s_0 and s_1). This notion of two-round OT can be obtained from LWE [BD18], or number theoretic assumptions such as DDH/QR and N$^{\text{th}}$ residuosity [NP01, AIR01, Kal05, HK12, DGI+19]. In summary, we prove the following theorem.

Theorem (informal). *There exists a 4-round MPC protocol that realizes any efficient functionality having black-box access to any two-round oblivious transfer protocol with statistical sender security.*

1.2 Technical Overview

The MPC in the head IPS compiler, and existing round efficient protocols. Our protocol is based on the approach proposed in [IKSS21] (and later improved in [IKSS23]), which is based on the so-called *IPS compiler* [IPS08] which, in turn, is based on the MPC-in-the-head framework of [IKOS07, IKOS09]. The IPS compiler relies on two main primitives: 1) an MPC protocol Φ (called *outer protocol*), secure in the honest majority setting tolerating adversarial parties that arbitrarily misbehave, and 2) a semi-honest protocol secure against a dishonest majority (called *inner protocol* that we denote with Π). In particular, the outer protocol Φ works in the client-server model, where n clients and m servers interact in two rounds of communication. In the first round, each client sends a message (which corresponds to a share of their input) to each of the m servers, and, in the second round, the servers send a message back to the clients which can now compute the output of the function. Φ is secure against malicious adversaries but requires the majority of the servers to be honest.

The goal of the IPS compiler is to combine the inner and outer protocol to exploit the benefits of both schemes: security against a dishonest majority and resiliency against adversaries misbehaving in an arbitrary way. Let f be a function that n parties want to securely evaluate. The high-level idea proposed in [IPS08] is to let the n parties run m (where m is polynomial in the security parameter) executions of the inner protocol where each of these executions simulates a server of Φ. Φ here is an outer protocol that realizes the function f. Combining the two protocols in this way clearly does not provide any additional security as nothing prevents the corrupted parties from misbehaving in all of the inner-protocol executions. Ishai et al. [IPS08] note that if one can argue that the adversarial parties behave correctly in the majority of the inner protocol executions, then it is possible to rely on the security of Φ, which, as we recall, is assumed to be secure in the honest-majority setting. To ensure that this is indeed the case (i.e., the corrupted parties behave correctly in the majority of the inner protocol executions), the IPS compiler requires each pair of parties to execute a cut-and-choose over the randomness-input pairs used in the inner-protocol executions via a so-called *watchlist* protocol. This watchlist protocol is realized by letting each pair of parties engage in an k-out-of-m Oblivious Transfer (OT). In each OT execution, one party acts as the sender using as its input the m defenses[6] (used in the m inner-protocol executions) and the receiver inputs a subset $K \subset \{1, \ldots, m\}$ of size k. Upon receiving the OT outputs, the receiver checks that all the defenses are valid, and, if this is the case, it proceeds with the execution of the protocol, otherwise, it aborts. The reason why this protocol is secure is that during the cut-and-choose, each receiver learns a number of shares that is insufficient to reconstruct the input of the other parties, but sufficient to argue whether the adversarial parties are behaving correctly in the majority of the inner protocol executions. This, in turn, means that if the cut-and-choose is successful, we can rely on the security of Φ.

In [IKSS21] the authors rely on a similar approach, but compress the IPS compiler into five rounds by devising an inner protocol that enjoys *delayed semi-malicious security* and a special type of four-round oblivious transfer protocol to realize the watchlist. Delayed semi-malicious security guarantees that security holds as long as the adversary provides valid defenses in the second last round of the protocol. Then, the authors instantiate the IPS compiler using the mentioned primitives to obtain a five-round protocol. The reason why the protocol does not have four rounds is mostly because the watchlist must be executed before the last round of the inner protocol is sent to the adversary (recall that the inner protocol is delayed semi-malicious). Hence, the watchlist is executed one round before the inner-protocol executions can terminate. The main takeaway from this is that if we want a four-round protocol, we need a three-round watchlist protocol. Indeed, this would make it possible to check that the first messages of the inner protocol are correct, allowing to rely on the security of the inner protocols that are completed during the last round of the overall protocol.

However, as mentioned in the first part of the introduction, non-trivial functionalities require at least four rounds of interaction to be realized. Given that the watchlist protocol realizes an OT-like functionality, it seems that we need different ideas if we want to go below 5 rounds. In [COSW23a], the authors show that the four-round lower bound can be actually circumvented by relaxing the ideal world OT functionality which, in turn, can be used to realize a three-round watchlist. The authors refer to this new functionality as *list OT functionality*, following [BGJ+18] that introduced a similar notion for the coin tossing functionality.

In this new OT definition, for a corrupted receiver, the ideal world is formalized as follows: the adversary provides its input $K \subset [m]$ and a parameter κ to the ideal functionality. The ideal functionality then samples κ random tuples representing the input of the sender $(s_1^i, s_2^i, \ldots, s_m^i)_{i \in \kappa}$, and provides $(s_j^i)_{i \in [\kappa], j \in K}$ to the adversary. The adversary now picks an index $c \in [\kappa]$ and sends it to the ideal functionality, which delivers $\{s_j^c\}_{j \in K}$ to the sender in the ideal world. This functionality can be seen as a random OT, in which the adversarial receiver can slightly bias the output of the computation.[7] After formalizing this new security definition, the authors of [COSW23a] show how to realize a three-round list OT protocol, that is secure against *sometimes aborting* adversaries, and generalize the result to obtain a three-round list watchlist protocol, secure in the same setting. At a high level, the notion of security

[6] A defense represents the input-randomness pair used to execute the inner protocol.

[7] This is not exactly a random OT as only the input of the honest senders is sampled randomly.

against sometimes aborting adversaries guarantees that if the adversary provides a valid third round, then it is possible to simulate, hence the inputs of the honest parties are protected. If, instead, the adversary aborts, then there are no security guarantees. If we now use this tool to replace the four-round watchlist protocol used in [IKSS21] then we obtain a final protocol that is secure only against sometimes aborting adversaries. This is the main reason why the protocol proposed in [COSW23a], which follows this paradigm, can only be proven secure against a restricted class of adversaries (or a restricted class of functionalities). We note that the reason why security does not hold against generic adversaries is because when the adversary is aborting, the senders' inputs of the watchlist protocol are leaked. Hence, all the randomness-input pairs that the honest parties used to run the inner protocol are accessible to the adversary and therefore the privacy of the honest parties is compromised. An alternative approach to obtain a three-round watchlist protocol has been proposed in [IKSS23]. Here, the authors show how to get security against any type of adversary relying on assumptions secure against sub-exponential time adversaries, thus obtaining a four-round protocol that is secure against any type of adversary. However, as just mentioned, this approach does not rely on polynomial time assumptions.

Our approach. Now, we are ready to discuss the approach taken in this work. Our approach is mostly based on the ideas of [COSW23a], i.e., we rely on a list watchlist protocol secure against sometimes aborting but we combine it with a special inner protocol. In particular, we observe that the three-round inner protocol proposed in [IKSS23, PS21] consists of two components: one component that samples some keys, and one component that, depending on the input of the parties, opportunely selects the keys, which are then used to compute the last message of the protocol. At a higher level, we can syntactically divide this inner protocol into two sub-algorithms. There is an interactive algorithm (denoted with $\Pi^{\mathsf{no\text{-}inp}}$) that on input some randomness provides correlated randomness, and another algorithm (denoted with Π^{inp}) that works by using only the input of the parties and the correlated randomness obtained from $\Pi^{\mathsf{no\text{-}inp}}$. We prove that this protocol retains its security under the condition that the adversary provides randomness that explains the messages generated using the sub-protocol $\Pi^{\mathsf{no\text{-}inp}}$. That is, the defense does not contain the input, but the simulator, upon receiving valid randomness, can extract the input by using the obtained randomness in combination with the messages generated from the adversary related to Π^{inp}. Equipped with this special inner-protocol, we can design the following variant of the IPS compiler.

We let the parties perform a cut-and-choose using the list watchlist protocol secure against sometimes aborting adversaries only over the randomness used to generate the messages of $\Pi^{\mathsf{no\text{-}inp}}$. This modification does not seem to add much security, as an adversary can still abort the list watchlist protocol, obtain the randomness of the honest parties, and use this randomness to infer the input of the honest parties looking at the messages generated by Π^{inp} (which are sent in the clear). To circumvent this issue, we rely on a conditional disclosure of secret (CDS), that allows the adversary to access the messages of Π^{inp} (the only messages encoding the inputs of the parties) only if it does not abort in the list watchlist protocol. To realize this CDS, we require each party to prepare a garbled circuit of the next message function of Π^{inp} parametrized with its input and the correlated randomness obtained from $\Pi^{\mathsf{no\text{-}inp}}$. The parties then send the garbled circuit in the last round of the overall protocol. To enable the evaluation of the garbled circuits, we additionally require the parties to engage in an execution of a standard simulation-based secure four-round OT protocol Π^{ot}. In particular, each party will act as a receiver in Π^{ot}, using as an input the message of Π^{inp}, and act as a sender in a second execution of Π^{ot} where the labels of the garbled circuit for the next-message function of Π^{inp} are used as an input. This allows the receiver to compute the last message of Π^{inp} and therefore the final output of the protocol. This mechanism allows to protect the messages generated using Π^{inp} as the adversary can obtain the messages related to Π^{inp} (by obtaining the garbled circuit and its labels in the fourth round) only if it successfully completes the third round. This implies that we disclose the messages of Π^{inp} only when it is ensured that the adversary behaves correctly in the majority of the inner-protocol executions, and, moreover, since the adversary did not abort enables us to rely on the security of the list watchlist protocol. On the other hand, if the adversary aborts during the execution of the list watchlist protocol, the receiver security of the OT protocol Π^{ot} (that enjoys the standard simulation-based security notion) guarantees that the messages of Π^{inp} generated by the honest parties remain protected.

Subtleties. The above high-level description omits a few subtleties. The first is that we are evaluating the next message function of Π^{inp} inside a garbled circuit, hence, we are making non-black-box use of Π^{inp}. To make sure that this does not cause an implicit non-black-box use of the primitives involved in the protocol, we will formally show that Π^{inp} does not make use of any computational cryptographic primitive. Indeed, we can move all the invocations to cryptographic primitives to the inputless part $\Pi^{\mathsf{no\text{-}inp}}$.

The second issue that arises is the pair-wise execution of a standard secure four-round OT protocol Π^{ot}. This allows a corrupted party to potentially use different inputs (i.e., different messages of Π^{inp}) in different OT executions, thus causing a security problem. Indeed, the security of the inner protocol holds under the condition that each party sends the same message to all the parties (i.e., each message is broadcast). To solve this issue, we replace the OT protocol Π^{ot} with a new primitive we call *Single Receiver, Multiple Senders Oblivious Transfer* ($\mathsf{1RnS}$ OT). This primitive guarantees that a receiver that engages in multiple OT executions against multiple senders uses the same input in all the executions. We instantiate our primitive using the techniques implicitly used in [CRSW22] (although for different reasons). We refer to Section 3 for more details on the security definition of $\mathsf{1RnS}$ and on how to instantiate it.

The last issue we need to solve is related to the fact that we are running two interactive primitives in parallel: the list watchlist protocol and our new $\mathsf{1RnS}$ OT protocol. This can cause rewinding issues. More precisely, the rewinds performed by the watchlist simulator may perturb a reduction to the receiver security of the OT. However, we note that when the adversary is aborting, we do not need to (and cannot) rely on the security of the watchlist protocol. Nevertheless, we still need to rely on the security of the $\mathsf{1RnS}$ OT protocol. On the other hand, when the adversary does not abort, we only need to rely on the security of the watchlist protocol as the messages of the inner protocol are guaranteed to be correctly generated. Therefore, no harm can be caused by an adversary that has access to the receivers' inputs of $\mathsf{1RnS}$ (that we recall, are the messages of Π^{inp}). For more details on how our final protocol works and how we deal with all these subtleties, we refer to the technical part of the paper.

2 Preliminaries

2.1 Oblivious Transfer

Private Two-Message Oblivious Transfer. A two-message oblivious transfer protocol consists of a tuple PPT algorithm $\mathsf{OT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ with the following syntax.

- $\mathsf{OT}_1(1^\lambda, \beta)$ takes the security parameter λ and a selection bit β and outputs a message ot_1 and secret state st.
- $\mathsf{OT}_2(1^\lambda, (\nu_0, \nu_1), \mathsf{ot}_1)$ takes the security parameter λ and two inputs $(\nu_0, \nu_1) \in \{0,1\}^{\mathsf{len}}$ (where len is a parameter of the scheme) and a message ot_1. It outputs a message ot_2.
- $\mathsf{OT}_3(1^\lambda, \mathsf{st}, \mathsf{ot}_2, \beta)$ takes the security parameter, the bit β, secret state st and message ot_2 and outputs $\nu_\beta \in \{0,1\}^{\mathsf{len}}$.

Correctness and security are defined as follows.

Definition 2.1 ([BD18]). *A tuple PPT algorithm* $\mathsf{OT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ *is a private-OT scheme id the following hold.*

Correctness *For all* $\lambda, \beta, \nu_0, \nu_1$, *letting* $(\mathsf{ot}_1, \mathsf{st}) = \mathsf{OT}_1(1^\lambda, \beta)$, $\mathsf{ot}_2 = \mathsf{OT}_2(1^\lambda, (\nu_0, \nu_1), \mathsf{ot}_1)$, $\nu' = \mathsf{OT}_3(1^\lambda, \mathsf{st}, \mathsf{ot}_2, \beta)$, *it holds that* $\nu' = \nu_\beta$ *with probability 1.*

Receiver Privacy *Consider the distribution* $\mathcal{D}_\beta(\lambda)$ *defined by running* $(\mathsf{ot}_1, \mathsf{st}) = \mathsf{OT}_1(1^\lambda, \beta)$ *and outputting* ot_1. *Then* $\mathcal{D}_0, \mathcal{D}_1$ *are computationally indistinguishable.*

Sender Privacy *There exists an (not necessarily efficient) extractor* OTExt *s.t. for any sequence of messages* $\mathsf{ot}_1 = \mathsf{ot}_1(\lambda)$ *and inputs* (ν_0, ν_1), *the distribution ensembles* $\mathsf{OT}_2(1^\lambda, (\nu_0, \nu_1), \mathsf{ot}_1)$ *and* $\mathsf{OT}_2(1^\lambda, (\nu_{\beta'}, \nu_{\beta'}), \mathsf{ot}_1)$, *where* $\beta' = \mathsf{OTExt}(\mathsf{ot}_1)$, *are statistically indistinguishable.*

Simulation-Based OT. Oblivious transfer (OT) is a two-party functionality F_{OT} in which a sender S holds a pair of strings (s_0, s_1), a receiver R holds a bit b and the receiver wants to obtain the string s_b. In this work, we focus on 4 round OT protocols with a delayed-input property. Delayed-input here means that the input of the receiver is only required in the third round and the input of the sender only in the fourth round.

Definition 2.2. *A delayed-input four-round OT protocol is a tuple of five algorithms* $\mathsf{OT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3, \mathsf{OT}_4, \mathsf{OT}_{\mathsf{out}})$ *with the following behavior:*

$\mathsf{OT}_1(1^\lambda; r_R)$: *This algorithm is executed by the receiver* R *and takes as an input a unary representation of the security parameter* 1^λ *and the randomness* r_R *of the receiver and outputs the first round message* ot_1.

$\mathsf{OT}_2(\mathsf{ot}_1; r_R)$**:** *This algorithm is executed by the sender S and takes as an input the message of the previous round* ot_1 *and the randomness r_S of the sender and outputs the second round message* ot_2*.*

$\mathsf{OT}_3(\{\mathsf{ot}_i\}_{i\in[2]}, b; r_R)$**:** *This algorithm is executed by the receiver R and takes as an input the previous round messages* $\{\mathsf{ot}_i\}_{i\in[2]}$*, the input bit b of the receiver and the randomness r_R of the receiver and outputs the third round message* ot_3*.*

$\mathsf{OT}_4(\{\mathsf{ot}_i\}_{i\in[3]}, (s_0, s_1); r_S)$**:** *This algorithm is executed by the sender S and takes as an input the previous round messages* $\{\mathsf{ot}_i\}_{i\in[3]}$*, the two input strings (s_0, s_1) of the receiver and the randomness r_S of the sender and outputs the fourth round message* ot_4*.*

$\mathsf{OT}_{\mathsf{out}}(\{\mathsf{ot}_i\}_{i\in[4]}; r_R)$**:** *This algorithm is executed by the sender R and takes as an input the previous round messages* $\{\mathsf{ot}_i\}_{i\in[4]}$ *and the randomness r_R of the receiver and computes the final output y.*

We say that a OT protocol is perfectly correct if for any $(s_0, s_1) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$ and any $b \in \{0,1\}$ it holds that:

$$\Pr[\mathsf{OT}_{\mathsf{out}}(\{\mathsf{ot}_i\}_{i\in[4]}; r_R) = s_b] = 1,$$

where $\mathsf{ot}_1 = \mathsf{OT}_1(1^\lambda; r_R)$*,* $\mathsf{OT}_2 = \mathsf{ot}_2(\mathsf{OT}_2; r_R)$*,* $\mathsf{ot}_3 = \mathsf{OT}_3(\{\mathsf{OT}_i\}_{i\in[2]}, b; r_R)$ *and* $\mathsf{ot}_4 = \mathsf{OT}_4(\{\mathsf{OT}_i\}_{i\in[3]}, (s_0, s_1); r_S)$ *for any $r_S, r_R \in \{0,1\}^\lambda$.*

In this work, we require one-sided simulation security, where we require the existence of a simulator against a malicious receiver and indistinguishability for the honest receiver's input. More formally:

Definition 2.3 (One-Sided Simulation [ORS15, COSV17]). *Let F_{OT} be the Oblivious Transfer functionality as described in 2.1. We say that a protocol Π securely computes F_{OT} with* one-sided simulation *if the following holds:*

1. *For every non-uniform PPT adversary R^* controlling the receiver in the real world, there exists a non-uniform PPT adversary* Sim *for the ideal world sucht that*

$$\{\mathrm{Real}_{\Pi,R^*(z)}(1^\lambda)\}_{z\in\{0,1\}*} \approx \{\mathrm{Ideal}_{F_{\mathsf{OT}},\mathsf{Sim}}(1^\lambda)\}_{z\in\{0,1\}*}$$

   where $\mathrm{Real}_{\Pi,R^*(z)}$ denotes the distribution of the output of the adversary R^* (controlling the receiver) after a real execution of the protocol Π where the sender S has inputs s_0, s_1 and the receiver has input b. $\mathrm{Ideal}_{F_{\mathsf{OT}},\mathsf{Sim}}$ denotes the analogous distribution in an ideal execution with a trusted party that computes F_{OT} for the parties and hands the output to the receiver.

2. *For every non-uniform PPT adversary S^* controlling the sender it holds that:*

$$\{\mathrm{View}^R_{\Pi,S^*(z)}(s_0, s_1, 0)\}_{z\in\{0,1\}*} \approx \{\mathrm{View}^R_{\Pi,S}(s_0, s_1, 1)\}_{z\in\{0,1\}*}$$

   where $\mathrm{View}^R_{\Pi,S^*(z)}$ denotes the view of adversary S^* after a real execution of Π with the honest receiver R.

Figure 2.1: The Oblivious Transfer Functionality F_{OT}

Functionality F_{OT}

F_{OT} running with a sender S, a receiver R and an adversary Sim proceeds as follows:

- Upon receiving a message $(\mathsf{send}, s_0, s_1, S, R)$ from S where each $s_0, s_1 \in \{0,1\}^\lambda$, record the tuple (s_0, s_1) and send send to R and Sim. Ignore any subsequent send messages.
- Upon receiving a message $(\mathsf{receive}, b)$ from R, where $b \in \{0,1\}$ send s_b to R and $\mathsf{receive}$ to S and Sim and halt. (If no $(\mathsf{send}, \cdot)$ message was previously sent, do nothing).

We also consider another OT functionality F^m_{OT} where the sender S has $2m$ inputs and the receiver R has as input a string of m bits; detail are provider below.

Definition 2.4 (Parallel Oblivious Transfer Functionality F^m_{OT} [ORS15, COSV17]). *The parallel Oblivious Transfer Functionality F^m_{OT} is identical to the functionality F_{OT}, with the difference that it takes as an input m pairs of strings $(s^1_0, s^1_1, \dots, s^m_0, s^m_1)$ from the sender S (whereas F_{OT} just takes one pair of strings from S) and m bits $(b_1, \dots, b_m)$ from R (whereas F_{OT} just takes one bit from R) and outputs $(s^1_{b_1}, \dots, s^m_{b_m})$ to the receiver while the sender receives nothing.*

Also in the case we define one-sided simulation security.

Definition 2.5 (One-Sided Simulation Security for parallel Oblivious Transfer [ORS15, COSV17]). *Let F^m_{OT} be the Oblivious Transfer functionality as described in Definition 2.4. We say that a protocol Π securely computes F^m_{OT} with* one-sided simulation *if the following holds:*

1. Sender Simulatability: *For every non-uniform PPT adversary R^* controlling the receiver in the real world, there exists a non-uniform PPT adversary Sim for the ideal world sucht that*
$$\{\mathrm{Real}_{\Pi, R^*(z)}(1^\lambda)\}_{z\in\{0,1\}*} \approx \{\mathrm{Ideal}_{F_{\mathsf{OT}},\mathsf{Sim}}(1^\lambda)\}_{z\in\{0,1\}*}$$
where $\mathrm{Real}_{\Pi, R^*(z)}$ denotes the distribution of the output of the adversary R^* (controlling the receiver) after a real execution of the protocol Π where the sender S has inputs $(s^1_0, s^1_1, \dots, s^m_0, s^m_1)$ and the receiver has inputs $(b_1, \dots, b_m)$. $\mathrm{Ideal}_{F_{\mathsf{OT}},\mathsf{Sim}^{r^*(z)}}$ denotes the analogous distribution in an ideal execution with a trusted party that computes F^m_{OT} for the parties and hands the output to the receiver.
2. Receiver Indistinguishability: *For every non-uniform PPT adversary S^* controlling the sender it holds that:*
$$\begin{aligned}&\{\mathrm{View}^R_{\Pi,S^*(z)}((s^1_0, s^1_1, \dots, s^m_0, s^m_1), (b_1, \dots, b_m))\}_{z\in\{0,1\}*}\\ \approx&\{\mathrm{View}^R_{\Pi,S^*(z)}((s^1_0, s^1_1, \dots, s^m_0, s^m_1), (b'_1, \dots, b'_m))\}_{z\in\{0,1\}*}\end{aligned}$$
where $\mathrm{View}^R_{\Pi,S^*(z)}$ denotes the view of adversary S^* after a real execution of Π with the honest receiver R for any $b_i \in \{0,1\}$ and $b'_i \in \{0,1\}$ for all $i \in [m]$.

In the rest of the paper whenever we refer to an OT protocol, we mean an OT protocol that implements the F^m_{OT} functionality. We note that the construction (Section 4) of [MOSV22] satisfies Definition 2.5 and makes black-box use of private oblivious transfer. Moreover, we note that the simulator $\mathsf{Sim}_{\mathsf{MOSV22}}$ (described to prove the sender simulatability properties in Section 4 of [MOSV22]) has the following simulation strategy against a malicious receiver R^*:

- $\mathsf{Sim}_{\mathsf{MOSV22}}$ upon receiving ot^1 from R^* produces ot^2.
- $\mathsf{Sim}_{\mathsf{MOSV22}}$ upon receiving ot^3 from R^*, proceeds to the extraction of the R^*'s input (rewinding R^* from the third to the second round).
- $\mathsf{Sim}_{\mathsf{MOSV22}}$ invokes the ideal functionality using the input extracted from R^*, and uses the output of the ideal functionality to produces ot^4.

In the rest of the paper, we will indicate a simulator with the above simulation strategy as a simulator that maintains the main thread.

2.2 MPC Definitions

Here, we provide a formal definition of secure multiparty computation verbatim taked from [IKSS21] which, in turn has been taken from [Ode09].

A multiparty protocol is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality. The security of a protocol is defined with respect to a functionality f. In particular, let n denote the number of parties. A non-reactive n-party functionality f is a (possibly randomized) mapping of n inputs ro n outputs. A multiparty protocol with security parameter λ for computing a non-reactive functionality f is a protocol running in time poly(λ) and satisfying the following correctness requirement: if parties $P_1, \ldots, P_n$ with inputs $(x_1, \ldots, x_n)$ respectively, all run an honest execution of the protocol, then the joint distribution of the outputs $y_1, \ldots, y_n$ of the parties is statistically close to $f(x_1, \ldots, x_n)$. A reactive functionality f is a sequence of non-reactive functionalities $f = (f_1, \ldots, f_\ell)$ computed in a stateful fashion in a series of phases. Let x_i^j denote the input of P_i in phase j, and let s^j denote the state of the computation after phase j. Computation of f proceeds by setting s^0 equal to the empty string and then computing $(y_1^j, \ldots, y_n^j, s^j) \leftarrow f_j(s^{j-1}, x_1^j, \ldots, x_n^j)$ for $j \in [\ell]$, where y_i^j denotes the output of P_i at the end of phase j. A multiparty protocol computing f also runs in ℓ phases, at the beginning of which each party holds an input and at the end of which each party obtains an output. (Note that parties may wait to decide on their phase-j input until the beginning of that phase.) Parties maintain state throughout the entire execution. The correctness requirement is that, in an honest execution of the protocol, the joint distribution of all the outputs $\{y_1^j, \ldots, y_n^j\}_{j=1}^{\ell}$ of all the phases is statistically close to the joint distribution of all the outputs of all the phases in a computation of f on the same inputs used by the parties.

Defining Security We assume that the reader is familiar with standard simulation-based definitions of security multiparty computation in the standalone setting. We provide a self-contained definition for completeness and refer to [Ode09] for a more complete description. The security of a protocol (w.r.t. a functionality f) is defined by comparing the real-world execution of the protocol with an ideal-world evaluation of f by a trusted party. More concretely, it is required that for every adversary $\mathcal{A}$, which attacks the real execution of the protocol, there exists an adversary Sim, also referred to as the simulator, which can *achieve the same effect* in the ideal-world. We denote $\vec{x} = (x_1, \ldots, x_n)$.

The real execution. In the real execution the n-party protocol π for computing f is executed in the presence of an adversary $\mathcal{A}$. The honest parties follow the instructions of π. The adversary $\mathcal{A}$ takes as input the security parameter λ, the set $I \subset [n]$ of corrupted parties, the inputs of the corrupted parties, and an auxiliary input z. $\mathcal{A}$ sends all messages in place of corrupted parties and may follow an arbitrary polynomial-time strategy.

The interaction of $\mathcal{A}$ with a protocol π defines a random variable $\mathrm{Real}_{\pi,\mathcal{A}(z),I}(\lambda, \vec{x})$ whose value is determined by the coin tosses of the adversary and the honest players. This random variable contains the output of the adversary (which may be an arbitrary function of its view) as well as the outputs of the uncorrupted parties. We let $\mathrm{Real}_{\pi,\mathcal{A}(z),I}$ denote the distribution ensemble $\{\mathrm{Real}_{\pi,\mathcal{A}(z),I}(\lambda, \vec{x})\}_{\lambda \in \mathbb{N}, \vec{x}, z \in \{0,1\}^*}$.

The ideal execution - security with abort. In this model, an ideal execution for a function f proceeds as follows:

- **Send inputs to the trusted party:** As before, the parties send their inputs to the trusted party, and we let x'_i denote the value sent by P_i.
- **Trusted party sends output to the adversary:** The trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends $\{y_i\}_{i \in I}$ to the adversary.
- **Adversary instructs trusted party to abort or continue:** This is formalized by having the adversary send either a continue or abort message to the trusted party. (A semi-honest adversary never aborts.) In the latter case, the trusted party sends to each uncorrupted party P_i its output value y_i. In the former case, the trusted party sends the special symbol $\perp$ to each uncorrupted party.
- **Outputs:** Sim outputs an arbitrary function of its view, and the honest parties output the values obtained from the trusted party.

The interaction of Sim with the trusted party defines a random variable $\mathrm{Ideal}_{f,\mathsf{Sim}(z)}(\lambda, \vec{x})$ as above, and we let $\{\mathrm{Ideal}_{f,\mathsf{Sim}(z),I}(\lambda, \vec{x})\}_{\lambda \in \mathbb{N}, \vec{x}, z \in \{0,1\}^*}$. Having defined the real and the ideal worlds, we can now proceed to define the security notion.

Definition 2.6. *Let λ be the security parameter, f an n-party randomized functionality, and Π an n-party protocol for $n \in \mathbb{N}$. We say that Π t-securely computes f in the presence of malicious adversaries if for every PPT adversary $\mathcal{A}$ there exists a PPT adversary* Sim *such that for any $I \subset [n]$ with $|I| \leq t$ the following quantity is negligible:*

$$|\Pr[\mathrm{Real}_{\pi,\mathcal{A}(z),I}(\lambda, \vec{x}) = 1] - \Pr[\mathrm{Ideal}_{f,\mathcal{A}(z),I}(\lambda, \vec{x}) = 1]|,$$

where $\vec{x} = \{x_i\}_{I \in [n]} \in \{0,1\}^*$ and $z \in \{0,1\}^*$.

Remark 2.7 (Security with Selective Abort). We can consider a slightly weaker definition of security where the ideal world adversary can instruct the trusted party to send aborts to a subset of the uncorrupted parties. For the rest of the uncorrupted parties, it instructs the trusted functionality to deliver their output. This weakened definition is called security with selective abort.

Remark 2.8 (Privacy with Knowledge of Outputs). Ishai et al. [IKP10] considered a further weakening of the security definition where the trusted party first delivers the output to the ideal world adversary which then provides an output to be delivered to all the honest parties. They called this security notion privacy with knowledge of outputs and showed a transformation from this notion to security with selective abort using unconditional MACs.

2.3 List MPC

The definitions here are taken from [COSW23b]. Let f be the function that n parties $P_1, \ldots, P_n$ want to compute: $f : X^0 \times \cdots \times X^n \to Y^1 \times \cdots \times Y^n$. We denote with $\mathcal{X}^i$ a PPT sampler for X^i for each $i \in [n]$. The notion of list MPC differs from the standard notion of MPC as follows. Let $M \subseteq [n]$ denote the indices of the corrupted parties and $H := [n] \setminus M$. In the ideal world experiment, the adversary sends its inputs $\{x^i \in \mathcal{X}^i\}_{i \in M}$ to the ideal functionality, which we denote with $\mathcal{F}^{\{\mathcal{X}^i\}_{i \in H}}$, together with an integer k. The ideal functionality does not wait to receive the input from the honest parties, instead, it samples k inputs uniformly at random from the input domains of the honest parties $\{x_1^i \leftarrow \mathcal{X}^i\}_{i \in H}, \ldots, \{x_k^i \leftarrow \mathcal{X}^i\}_{i \in H}$, and, for each $j \in [k]$ computes $(\mathsf{out}_j^1, \ldots, \mathsf{out}_j^n) \leftarrow f(\{x_j^i\}_{i \in H}, \{x_j^i\}_{i \in M})$. Then the values $\{\mathsf{out}_1^i, \ldots, \mathsf{out}_k^i\}_{i \in M}$ are given to the adversary. The adversary now sends an index $j \in [k]$ to the ideal functionality, which then sends $(x_j^i, \mathsf{out}_j^i)$ to the ideal-world honest party P_i for each $i \in H$. We propose the formal definition of the real and the ideal world in Figure 2.2. To make this definition stronger and usable as a sub-routine of other protocols, the adversary is given the chance to see the inputs of the honest parties.[8] In the ideal world, we do that by explicitly giving this input to the adversary together with the second round of the protocol. In the ideal world experiment, we do something similar, by allowing to send the input of the honest parties to the adversary upon request of the simulator. More precisely, if the simulator sends a message to the adversary $(\{\pi_i\}_{i \in H}, j)$, then the real-world experiment would forward to the adversary the message $(\{\pi_i, x_j^i\}_{i \in H})$. This guarantees that the distinguisher will get access to the honest party's input, while the simulator does not (this is exactly what makes our definition non-trivial to realize). We refer to Figure 2.2 for the formal specification of real and ideal world and state the following:

Definition 2.9 (List MPC). *We say that a protocol Π is a secure List MPC protocol if for every malicious PPT adversary $\mathcal{A}$ corrupting an arbitrary set of parties in indices $M \subseteq [n]$ (the indices of the honest parties are denoted with $H := [n] \setminus M$), there exists a (expected) PPT simulator* $\mathsf{Sim} = \{\mathsf{Sim}_1, \mathsf{Sim}_2\}$ *such that*

$$\{\mathsf{Real}_{\mathcal{A},\Pi}(1^\lambda, M, H)\}_\lambda \approx_c \{\mathsf{Ideal}_{\mathsf{Sim},\mathcal{F}^{\{\mathcal{X}^i\}}}(1^\lambda, M, H)\}_\lambda,$$

[8] In the standard MPC definition the inputs of the honest parties are fixed from the beginning, hence the adversary and the distinguisher are implicitly aware of the honest parties inputs.

where $\lambda \in \mathbb{N}$.

Figure 2.2: Real and ideal world

$\mathsf{Real}_{\mathcal{A},\Pi}(1^\lambda, M, H)$:

1. For each $i \in H$ sample $x_i \leftarrow \mathcal{X}^i$, and compute the first round π_1^i of Π as the honest P_i would do on input x_i and the randomness $r_i \leftarrow \{0,1\}^\lambda$ and send π_1^i to $\mathcal{A}$.
2. Upon receiving $\{\pi_1^i\}_{i \in M}$ from $\mathcal{A}$, for each $i \in H$ compute the second round π_2^i of Π as the honest party P_i on input (r, x_i) would do and send (π_2^i, x_i) to $\mathcal{A}$.
3. Upon receiving $\{\pi_2^i\}_{i \in M}$ from $\mathcal{A}$, for each $i \in H$ compute π_3^i as the honest party P_i would do having on input $(\pi_1^i, \pi_2^i, r_i, x_i)$ and send π_3^i to $\mathcal{A}$.
4. Upon receiving $\{\pi_3^i\}_{i \in M}$, compute the output out_i as P_i would do, and return the view of $\mathcal{A}$.

$\mathsf{Ideal}_{\mathsf{Sim},\mathcal{F}^{\{\mathcal{X}^i\}_{i \in M}}}(1^\lambda, M, H)$:

1. $(\{x^i\}_{i \in M}, 1^k, z) \leftarrow \mathsf{Sim}_1^{\mathcal{A}}(1^\lambda)$
2. Send $\{x^i\}_{i \in M}$ to the ideal functionality $\mathcal{F}^{\{\mathcal{X}^i\}_{i \in M}}$ which computes $\{\mathsf{out}_j^i\}_{i \in M, j \in [k]}$ as described using the sampled inputs $\{x_j^i\}_{j \in [k]}$.
3. Whenever $\mathsf{Sim}_2^{\mathcal{A}}(\{\mathsf{out}_j^i\}_{i \in M, j \in [k]}, z)$ queries $\mathcal{A}$ with a message $(\{\pi_i\}_{i \in H}, j)$, replace the query with $(\{\pi_i, x_j^i\}_{i \in H})$ and forward the pair to $\mathcal{A}$.
4. $(i, \text{View}) \leftarrow \mathsf{Sim}_2^{\mathcal{A}}(\{\mathsf{out}_j^i\}_{i \in M, j \in [k]}, z)$ with $i \in [k] \cup \{\bot\}$.
5. Send i to the ideal functionality, or abort if $i = \bot$ to instruct the honest party to abort, and return View.

Sometimes aborting adversaries. In this paper we consider the notion of sometimes aborting adversaries. This notion is the same as the list simulation notion, but it requires the list simulator to provide a simulated transcript with overwhelming probability, conditioned on the adversary providing an accepting transcript in the main thread. We still provide no security requirements (unless otherwise specified), in the case that the adversary aborts with overwhelming probability, or it does not provide an accepting transcript in the main thread.

Multiparty Simultaneous OT. Now, we recap the definition for the list (multiparty) simultaneous OT functionality which has been introduced in [COSW23a]. In this work, the authors also show how to realize this primitive from statistical sender private two-message OT. The (multiparty) simultaneous OT functionality is an n-party functionality that implements simultaneous (or parallel) α-out-of-β OTs between n parties. Informally, this functionality consists of $n \cdot (n-1)$ instances of α-out-of-β OT. For every $i \in [n], j \in [n], j \neq i$, a secure OT is implemented between the pair (P_i, P_j) where P_i is the sender and P_j is the receiver.

Formally, we define the ideal (multiparty) simultaneous OT functionality $\mathcal{F}_{\mathsf{OT}}^{n \cdot (n-1)}$. this consists of $n(n-1)$ independent instances of a α-out-of-β OTs, one for each ordered pair $(i,j) \in [n] \times [n]$ such that $i \neq j$. The (i,j)-th OT instance obtains input $x_{i,j} = (x_{i,j}^1, \ldots, x_{i,j}^\beta)$ form sender P_i and input $y_{i,j} \subseteq [\beta]$ with $|y_{i,j}| = \alpha$ from the receiver P_j. The functionality then outputs $\{x_{i,j}^k\}_{k \in y_{j,i}}$ to P_j and outputs $\bot$ to P_i for each $i, j \in \{[n] \times [n]\}$ with $i \neq j$.

Definition 2.10 (Simultaneous OT Protocol). *A (multiparty) simultaneous OT protocol* mpOT *is defined by a tuple of algorithms* $(\mathsf{mpOT}_1, \mathsf{mpOT}_2, \mathsf{mpOT}_3, \mathsf{out}_{\mathsf{mpOT}})$*. For each round* $r \in [4]$*, the* i*-th party in the protocol runs* mpOT_r *on* 1^λ*, the index* i*, the private input* $\{x_{i,j}, y_{i,j}\}_{i \neq j}$ *and the transcript of the protocol in the first* $(r-1)$ *rounds to obtain* mpOT_r^i*. It sends* mpOT_r^i *to every other party via a broadcast channel. We let* $\mathsf{mpOT}(r)$ *denote the transcript of the protocol* mpOT *in the first* r *rounds. The output computing function* $\mathsf{out}_{\mathsf{mpOT}}$ *takes the index* i *of a party, its private input, its random tape, and the transcript* $\mathsf{mpOT}(3)$ *and generates the output* out_i*. The protocol is required to satisfy the following property correctness* *property* $\Pr[(\mathsf{out}_1, \ldots, \mathsf{out}_n) = \mathcal{F}_{\mathsf{OT}}^{n\cdot(n-1)}(\{x_{i,j}, y_{i,j}\}_{i\in[n],j\neq i})] = 1$*, where* out_i *denotes the output obtained by the* i*-th party from the* mpOT *protocol when executed on the private input and random tape of* P_i*.*

In terms of security, we consider a slight variation of the notion of list MPC that we have recalled in Definition 2.9. In particular, in this notion we have two types of inputs: *static*, and *list*. The static inputs are inputs decided at the onset of the ideal/real-world experiment, whereas the list-inputs are the inputs sampled by the ideal functionality (on behalf of the honest parties) by using some samplers $\{\mathcal{X}_i\}_{i\in[H]}$. In the specific case of the OT functionality we consider here, the inputs of the honest receivers are fixed, whereas the inputs of the honest senders are sampled from the ideal functionality in the spirit of Definition 2.9. We provide the formal definition of the ideal and real-world in Figure 2.4, and we say that a simultaneous OT protocol is secure if it satisfies the following.

Definition 2.11 (Secure Simultaneous OT Protocol). *Let* $\mathcal{A}$ *be an adversary corrupting an arbitrary subset of parties indexed by* M *(we denote the indices of the honest parties with* $H = [n] \setminus M$*), that obtains auxiliary input* aux*. Let* $\{\mathsf{Real}_{\mathcal{A},\mathsf{mpOT}}(1^\lambda, M, H, \{y_{i,j}\}_{i\in H,j\in M})\}_\lambda$ *denote the joint distribution of the view of the adversary and the outputs of honest parties in the real-world world experiment described in Figure 2.4. We require the existence of an expected polynomial time simulator* Sim *that with black-box access to the adversary* $\mathcal{A}$ *interacts with the ideal functionality* $\mathcal{F}^{\mathsf{list}}_{OT^{\alpha,\beta}}$ *(see Figure 2.3) as described in the ideal world experiment of Figure 2.4, and return an output, denoted by* $\{\mathsf{Ideal}_{\mathsf{Sim},\mathcal{F}^{\mathsf{list}}_{OT^{\alpha,\beta}}}(1^\lambda, M, H, \{y_{i,j}\}_{i\in H,j\in M})\}$ *such that the following holds for any* $\{y_{i,j}\}_{i\in H,j\in M}$ $\{\mathsf{Real}_{\mathcal{A},\mathsf{mpOT}}(1^\lambda, M, H, \{y_{i,j}\}_{i\in H,j\in M})\}_{\lambda\in\mathbb{N}}$ *is indistinguishable from* $\{\mathsf{Ideal}_{\mathsf{Sim},\mathcal{F}^{\mathsf{list}}_{OT^{\alpha,\beta}}}(1^\lambda, M, H, \{y_{i,j}\}_{i\in H,j\in M})\}_{\lambda\in\mathbb{N}}$.

Figure 2.3: The list functionality $\mathcal{F}^{\mathsf{list}}_{OT^{\alpha,\beta}}$

The functionality is parametrized by the samplers $\{\mathcal{X}_i\}_{i\in H}$, and by the honest receivers inputs $\{y_{i,j}\}_{i\in H,j\in M}$, where $y_{i,j} \subseteq [\beta]$, with $|y_{i,j}| = \alpha$ for each $i \in H, j \in M$.

1. Upon receiving the receivers-inputs of the corrupted parties $\{y_{i,j}\}_{i\in M,j\in H}$, and $k \in \mathbb{N}$, for each $\kappa \in [k]$ do the following
   - For each $i \in H, j \in M$ sample $x_{i,j,\kappa} \leftarrow \mathcal{X}_i$, where $x_{i,j,\kappa} = (x^1_{i,j,\kappa}, \ldots, x^\beta_{i,j,\kappa})$, and compute $\mathsf{out}^\kappa_{i,j} := \{x^c_{i,j,\kappa}\}_{c\in y_{j,i}}$
2. Send $\{\mathsf{out}^\kappa_{i,j}\}_{\kappa\in[k],i\in H,j\in M}$ to $\mathcal{A}$
3. Upon receiving $(\iota, \{x_{i,j}\}_{i\in M,j\in H}, \mathsf{abort})$ from $\mathcal{A}$ do the following. For each $j \in H \setminus \mathsf{abort}$ send $\{x^c_{i,j}\}_{i\in M,c\in y_{j,i}}$ and $\{x_{j,i,\iota}\}_{i\in M}$ to the honest P_j. For each $j \in \mathsf{abort}$ send an abort command to P_j.

Figure 2.4: Real and ideal world

$\mathsf{Real}_{\mathcal{A},\mathsf{mpOT}}(1^\lambda, M, H, \{y_{i,j}\}_{i\in H, j\in M})$

1. For each $i \in H$ sample $\{x_{i,j}\}_{j\in[M]} \leftarrow \mathcal{X}^i$.
2. For each $i \in H$ compute the first round π_1^i of mpOT as the honest P_i would do on input $(\{x_{i,j}, y_{i,j}\}_{j\in[n]\setminus\{i\}})$ and the randomness $r_i \leftarrow \{0,1\}^\lambda$ and send π_1^i to $\mathcal{A}$.
3. Upon receiving $\{\pi_1^i\}_{i\in M}$ from $\mathcal{A}$, for each $i \in H$ compute the second round π_2^i of mpOT as the honest party P_i thus obtaining π_2^i and send $(\pi_2^i, \{x_{i,j}\}_{j\in M})$ to $\mathcal{A}$.
4. Upon receiving $\{\pi_2^i\}_{i\in M}$ from $\mathcal{A}$, for each $i \in H$ compute π_3^i as the honest party P_i would do thus obtaining π_3^i, and send π_3^i to $\mathcal{A}$.
5. Upon receiving $\{\pi_3^i\}_{i\in M}$, compute the output out_i as P_i would do, and return the view of $\mathcal{A}$.

$\mathsf{Ideal}_{\mathsf{Sim},\mathcal{F}^{\mathsf{list}}_{OT^{\alpha,\beta}}}(1^\lambda, M, H, \{y_{i,j}\}_{i\in H, j\in M})$

1. $(\{y_{i,j}\}_{i\in M, j\in H}, 1^k, z) \leftarrow \mathsf{Sim}_1^{\mathcal{A}}(1^\lambda)$
2. Send $\{y_{i,j}\}_{i\in M, j\in H}$ to the ideal functionality $\mathcal{F}^{\mathsf{list}}_{OT^{\alpha,\beta}}$ which computes $\{\mathsf{out}^\kappa_{i,j}\}_{i\in H, j\in M, \kappa\in[k]}$ as described using the sampled inputs $\{x_{i,j,\kappa}\}_{i\in[H], j\in M, \kappa\in[k]}$ as described before.
3. Whenever $\mathsf{Sim}_2^{\mathcal{A}}(\{\mathsf{out}^\kappa_{i,j}\}_{i\in M, j\in H, \kappa\in[k]}, z)$ queries $\mathcal{A}$ with a message $(\{\pi_i\}_{i\in H}, \gamma)$ (with $\gamma \in [k]$), replace the query with $(\{\pi_i\}_{i\in H}, \{x_{i,j,\gamma}\}_{i\in[H], j\in M}, \gamma)$ and forward the pair to $\mathcal{A}$.
4. $(\iota, \{x_{i,j}\}_{i\in[M], j\in H}, \mathrm{View}) \leftarrow \mathsf{Sim}_2^{\mathcal{A}}(\{\mathsf{out}^\kappa_{i,j}\}_{i\in M, j\in H, \kappa\in[k]}, z)$ with $\iota \in [k] \cup \{\bot\}$.
5. Send $(\iota, \{x_{i,j}\}_{i\in[M], j\in H}, \mathsf{abort})$ where abort is a set of indices that determines which honest parties should not receive the output from $\mathcal{F}^{\mathsf{list}}_{OT^{\alpha,\beta}}$.
6. Return View.

2.4 Garbled Circuits

This section is taken verbatim from [PS21].

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

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

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

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

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

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

$$(\widetilde{C}, \{\mathsf{lab}_{w,x_w}\}_{w\in\mathsf{inp}(C)}) \approx_c \mathsf{Sim}_{\mathsf{GC}}(1^{|C|}, 1^{|x|}, C(x))$$

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

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

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

As noted in [GS18], the authenticity of input labels property can be added to any garbled circuit by digitally signing the labels and including the signatures along with the labels and including the verification key along with the garbled circuit $\widetilde{C}$. The evaluation procedure first checks the signatures before proceeding with the actual evaluation of the garbled circuit.

2.5 Symmetric Encryption and Message Authentication

In this section, we recall the definitions of symmetric encryption, message authentication codes, digital signatures, and commitments.

Definition 2.12 (Symmetric Encryption [GB96]). *A symmetric encryption scheme (SE) for the message space $\mathcal{M}$ is a tuple of three algorithms $\mathsf{SE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$:*

$\mathsf{Setup}(1^\lambda)$**:** *Takes as input a unary representation of the security parameter λ, and outputs a key k.*

$\mathsf{Enc}(\mathsf{k}, m)$**:** *Takes as input the symmetric key k, a message $m \in \mathcal{M}$ to encrypt, and outputs a ciphertext ct.*

$\mathsf{Dec}(\mathsf{k}, \mathsf{ct})$**:** *Takes as input the symmetric key k and a ciphertext ct and outputs a message or $\perp$ if decryption fails.*

A scheme SE is correct, if for all $\lambda \in \mathbb{N}$, $\mathsf{k} \leftarrow \mathsf{Setup}(1^\lambda)$, $m \in \mathcal{M}$, we have

$$\Pr\left[\mathsf{Dec}(\mathsf{k}, \mathsf{Enc}(\mathsf{k}, m)) = m\right] = 1 \ .$$

Security for a symmetric encryption scheme is defined in an indistinguishable manner.

Definition 2.13 (IND-CPA Security of SE). *Let $\mathsf{SE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ be an SE scheme, for the message space $\mathcal{M}$. For $\beta \in \{0,1\}$, we define the experiment $\text{IND-CPA}_\beta^{\mathsf{SE}}$ in Fig. 1, where the encryption oracle QEnc outputs $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{k}, x_\beta)$ on a query (x_0, x_1). We define the advantage of an adversary $\mathcal{A}$ in the following way*

$$\mathsf{Adv}_{\mathsf{SE},\mathcal{A}}^{\text{IND-CPA}}(\lambda) = |\Pr[\text{IND-CPA}_0^{\mathsf{SE}}(\lambda, \mathcal{A}) = 1] - \Pr[\text{IND-CPA}_1^{\mathsf{SE}}(\lambda) = 1]| \ .$$

A symmetric encryption scheme SE is called IND-CPA secure, if for any PPT adversary $\mathcal{A}$ it holds that $\mathsf{Adv}_{\mathsf{SE},\mathcal{A}}^{\text{IND-CPA}}(\lambda) \leq \text{negl}(\lambda)$.

```
$\mathbf{IND\text{-}CPA}^{\mathsf{SE}}_{\beta}(1^{\lambda}, \mathcal{A})$
$\mathsf{sk} \leftarrow \mathsf{Setup}(1^{\lambda})$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QEnc}(\cdot,\cdot)}(1^{\lambda})$
**Output:** $\alpha$
```

Fig. 1: IND-CPA Security Game for a symmetric encryption scheme SE.

In our protocol, it is sufficient to use a symmetric encryption scheme that fulfills *one-time security*. In this special case of IND-CPA security the encryption oracle can only be queried once. The one-time pad is a candidate scheme that fulfills this notion.

Definition 2.14 (Message Authentication Code [Gol04]). *A message authentication code (MAC) for the message space $\mathcal{M}$ is a tuple of three algorithms* $(\mathsf{Setup}, \mathsf{Auth}, \mathsf{Verify})$*:*

$\mathsf{Setup}(1^{\lambda})$**:** *Takes as input a unary representation of the security parameter 1^{λ}, and outputs a key* k.
$\mathsf{Auth}(\mathsf{k}, m)$**:** *Takes as input the key* k*, a message $m \in \mathcal{M}$, and outputs a tag τ.*
$\mathsf{Verify}(\mathsf{k}, m)$**:** *takes as input a key* k*, a message m and a tag τ and outputs either* 0 *or* 1.

A scheme MAC *is correct, if for all $\lambda \in \mathbb{N}$,* $\mathsf{k} \leftarrow \mathsf{Setup}(1^{\lambda})$*, $m \in \mathcal{M}$, we have*

$$\Pr\left[\mathsf{Verify}(\mathsf{k}, m, \mathsf{Auth}(\mathsf{k}, m)) = 1\right] = 1 \ .$$

Definition 2.15 (Unforgeability of MAC). *Let* $\mathsf{MAC} = (\mathsf{Setup}, \mathsf{Auth}, \mathsf{Verify})$ *be a MAC, for the message space $\mathcal{M}$. We define the experiment* $\text{EUF-CMA}^{\mathsf{MAC}}$ *in Fig. 2 with Q being the set containing the queries of $\mathcal{A}$ to the authentication oracle* $\mathsf{Auth}(\mathsf{k}, \cdot)$.

A message authentication code MAC *is called existentially unforgeable under adaptive chosen-message attacks (EUF-CMA secure), if for any PPT adversary $\mathcal{A}$ it holds that* $\Pr[\text{EUF-CMA}^{\mathsf{MAC}}(\lambda, \mathcal{A}) = 1] \leq \text{negl}(\lambda)$.

```
$\mathbf{EUF\text{-}CMA}^{\mathsf{MAC}}(1^{\lambda}, \mathcal{A})$
$\mathsf{k} \leftarrow \mathsf{Setup}(1^{\lambda})$
$(m^*, \tau^*) \leftarrow \mathcal{A}^{\mathsf{Auth}(\mathsf{k},\cdot)}(1^{\lambda})$
**Output:** $\mathsf{Verify}(\mathsf{k}, m^*, \tau^*) = 1 \land m \notin Q$
```

Fig. 2: The Existentially Unforgeability Game for a message authentication code MAC.

2.6 Outer Protocol

The outer protocol that is used in this work needs to achieve the same properties as the outer protocol used in [IKSS21] which we recap here verbatim.

Their outer protocol is a 2-round, n-client, m-server MPC protocol achieving privacy with knowledge of outputs (Remark 2.8) against a malicious, adaptive adversary corrupting up to $n-1$ clients and $t = (m-1)/3$ servers. Such a protocol was constructed in [IKP10] making black-box use of a pseudorandom generator (PRG). We set $m = 8\lambda n^2$. We now give details about the syntax of this protocol:

1. In the first round, the i'th client runs Φ_1 on input 1^λ, the index i and its private input x_i to obtain $(\phi_1^{i\to 1}, \dots, \phi_1^{i\to m})$. Here, $\phi_1^{i\to j}$ denotes the private message that this client needs to send to the j'th server (for each $j \in [m]$) in the first round.
2. In the second round, the j'th server runs $\Phi_2(j, (\phi_1^{1\to j}, \dots, \phi_1^{n\to j}))$ to obtain ϕ_2^j and this is sent to the output client in the second round.
3. Finally, the output client runs $\mathsf{out}_\Phi(\phi_2^1, \dots, \phi_2^m)$ to compute the output of the protocol.

Remark 2.16. We require the functions computed by the servers to be information-theoretic and not involve any cryptographic operations. In the protocol of [IKP10], the servers have to perform several PRG computations. To deal with this challenge, we delegate the computation of the PRG to each of the clients. Specifically, for every PRG computation to be done by each server, every client chooses a random seed and gives the output of the PRG on this seed to the server. Let $(\mathsf{seed}_i, \mathsf{PRG}(\mathsf{seed}_i))$ be the contribution from the i'th client where PRG has a sufficiently long stretch. The server sets $\mathsf{seed} = (\mathsf{seed}_1, \dots, \mathsf{seed}_n)$ and defines a new $\mathsf{PRG}'(\mathsf{seed}) = \oplus_i \mathsf{PRG}(\mathsf{seed}_i)$. The seed and the PRG computation is sent as part of the first message from the client to the servers. The watchlist protocol is then used to ensure that the PRG computations done by the honest servers are correct.

3 Single Receiver, Multiple Senders Oblivious Transfer (1RnS OT)

A 1RnS OT is a protocol between n parties acting as senders $\vec{S} = \{S_i\}_{i\in[n]}$ and one party acting as a receiver R. At a high level, the functionality realized by a 1RnS OT protocol can be described as follows: each sender engages in a 1-out-of-2 oblivious transfer protocol against the same receiver, which is guaranteed to use the same input, in all the n 1-out-of-2 OT executions. Formally, we denote the functionality realized by our protocol with $\mathcal{F}_{\vec{S},R}$, and denote the input of the i-th sender S_i, for each $i \in [n]$, with $(\mathbf{s}_i^0, \mathbf{s}_i^1)$, where $\mathbf{s}_i^d$ (with $d \in \{0,1\}$) is a vector that contains ℓ bit-strings (of size λ): $s_{i,1}^d, \dots, s_{i,\ell}^d$. We denote the receiver's input with $m \in \{0,1\}^\ell$. Upon receiving the inputs from the honest senders and the honest receiver, the functionality returns $(\{s_{i,1}^{m_1}\}_{i\in[n]}, \dots, \{s_{i,\ell}^{m_\ell}\}_{i\in[n]})$ to the receiver, and nothing to the senders. We say that a protocol is a secure 1RnS OT if it satisfies Definition 3.1 described below.

Definition 3.1. *An* 1RnS *OT protocol is defined by the following tuple of algorithms* $\mathsf{OT}_{\vec{S},R} = (\mathsf{OT}_{R,1}, \mathsf{OT}_{S,2}, \mathsf{OT}_{R,3}, \mathsf{OT}_{S,4}, \mathsf{OT}_{\mathsf{out}})$*, where the senders speak in the second and fourth rounds while the receiver speaks in the first and third rounds over a broadcast channel. We denote with* τ_{OT} *the transcript of the protocol. The algorithm* $\mathsf{OT}_{\mathsf{out}}$ *takes as input* τ_{OT}*, the input and randomness of the receiver and generates the output* out_R *for the receiver. We say that an* 1RnS *OT protocol is secure if it satisfies the following properties.*

Delayed-Input Correctness: *For every possible input* $(\{(\mathbf{s}_i^0, \mathbf{s}_i^1)\}_{i\in[n]}, m)$*, let* $\vec{S} = \{S_i\}_{i\in[n]}$ *be senders that follow the protocol specification, which receive the inputs at the end of the third round and, R be an honest receiver R which receives the input at the end of the second round, we have that:* $\Pr\left[\mathsf{out}_R = \mathcal{F}_{\vec{S},R}(\{(\mathbf{s}_i^0, \mathbf{s}_i^1)\}_{i\in[n]}, m)\right] = 1$*, where* out_R *denotes the output obtained by the receiver at the end of the protocol.*

Sender Privacy: *Let* $\mathcal{A}$ *be a PPT adversary, with any auxiliary input* z*, that corrupts the receiver and an arbitrary subset of senders with indices* $\mathcal{I} \subseteq [n]$*. For every set of vector pairs* $\{(\mathbf{s}_i^0, \mathbf{s}_i^1)\}_{i\in\mathcal{H}}$ *(where* $\mathcal{H} = \{1, \ldots, n\} \setminus \mathcal{I}$*) we require that there exists an expected polynomial time simulator* Sim *with the following characteristics.*

1. *It has black-box access to* $\mathcal{A}$*.*
2. *It works in two phases. In the first phase, it returns* m *(which implicitly denotes the input of the corrupted receiver). In the second phase, the simulator is fed with the vectors* $(\{\boldsymbol{\sigma}_i^0, \boldsymbol{\sigma}_i^1\}_{i\in\mathcal{H}}$ *constructed as follows. For each* $i \in [n], d \in \{0,1\}$ $\boldsymbol{\sigma}_i^d := (\sigma_{i,1}^d, \ldots, \sigma_{i,\ell}^d)$*, where for each* $i \in \mathcal{H}, j \in [\ell]$ $\sigma_{i,j}^{m_j} := s_{i,j}^{m_j}, \sigma_{i,j}^{1-m_j} := 0^\lambda$*.*
3. *Upon receiving the above input, the simulator returns the output and stops.*

Sender privacy requires that the following two distributions to be computationally indistinguishable: $\mathrm{View}(\mathcal{A}(z), \{S_i\}_{i\in\mathcal{H}}(\{\mathbf{s}_i^0, \mathbf{s}_i^1\}_{i\in\mathcal{H}})) \approx$ *and* $\{\mathsf{Sim}^{\mathcal{A}}(1^\lambda, z)\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^\star}$*, where* $\mathrm{View}_{\mathcal{A}}(\mathcal{A}(z), \{S_i\}_{i\in\mathcal{H}}, \{\mathbf{s}_i^0, \mathbf{s}_i^1\}_{i\in\mathcal{H}})$ *denotes the view of* $\mathcal{A}$ *during the execution of* $\mathsf{OT}_{\vec{S},R}$ *where the honest parties* $\{S_i\}_{i\in\mathcal{H}}$ *use the inputs* $\{(\mathbf{s}_i^0, \mathbf{s}_i^1)\}_{i\in\mathcal{H}}$*, and* $\mathsf{Sim}^{\mathcal{A}}(1^\lambda, z)$ *denotes the output of the simulator described above.*

Receiver Privacy: *Let* $\mathcal{A}$ *be a PPT adversary, with any auxiliary input* z*, that corrupts an arbitrary subset of senders with indices* $\mathcal{I} \subseteq [n]$*. For every set of vector pairs* $\{(\mathbf{s}_i^0, \mathbf{s}_i^1)\}_{i\in\mathcal{H}}$ *(where* $\mathcal{H} = \{1, \ldots, n\} \setminus \mathcal{I}$*) and every pair of* ℓ*-bit strings* m, m'*, we require the following to be computationally indistinguishable:* $\mathrm{View}(\mathcal{A}(z), (\{S_i\}_{i\in\mathcal{H}}(\{\mathbf{s}_i^0, \mathbf{s}_i^1\}_{i\in\mathcal{H}}), R(m)))$, $\mathrm{View}(\mathcal{A}(z), (\{S_i\}_{i\in\mathcal{H}}(\{\mathbf{s}_i^0, \mathbf{s}_i^1\}_{i\in\mathcal{H}}), R(m')))$*, where* $\mathrm{View}_{\mathcal{A}}(\mathcal{A}(z), (\{S_i\}_{i\in\mathcal{H}}, \{\mathbf{s}_i^0, \mathbf{s}_i^1\}_{i\in\mathcal{H}}, R(m)))$ *denotes the view of* $\mathcal{A}$ *during the execution of* $\mathsf{OT}_{\vec{S},R}$ *where the honest sender* $\{S_i\}_{i\in\mathcal{H}}$ *use the inputs* $\{(\mathbf{s}_i^0, \mathbf{s}_i^1)\}_{i\in\mathcal{H}}$ *and the receiver uses the input* m*.*

3.1 1RnS Oblivious Transfer from 1-out-of-2 Oblivious Transfer

In this section, we show how to obtain a $\mathsf{1RnS}$ OT protocol according to Definition 3.1. For simplicity, we consider the case where the receiver has one-bit input and each sender S_i has two strings as its input (s_i^0, s_i^1), for $i \in [n]$. Each sender S_i computes an n-out-of-n additive secret sharing of its inputs, obtaining $s_{i,1}^b, \ldots, s_{i,n}^b$ such that $s_i^b = s_{i,1}^b \oplus \cdots \oplus s_{i,n}^b$ for each $b \in \{0,1\}$. Then, S_i sends the shares $(s_{i,j}^0, s_{i,j}^1)$ to S_j for each $i \neq j$ privately.[9]

In parallel, each sender S_i engages with the receiver in an execution of a 1-out-of-2 simulation-based secure four-round OT, which we denote with OT, using, as its first input, $(s_{1,i}^0||\ldots||s_{n,i}^0)$, and $(s_{1,i}^1||\ldots||s_{n,i}^1)$ as its second input. The honest receiver uses the same input m in all the executions of OT. At the end of the protocol, the honest receiver can reconstruct the outputs by computing $s_{i,1}^m \oplus \cdots \oplus s_{i,1}^m$ for each $i \in n$. The above mechanism

[9] The private communication is implemented using CPA public-key encryption, where the keys are exchanged in the second round and the shares are encrypted and sent in the third round.

ensures that if a corrupted receiver uses different inputs across different executions of OT, that it will not be able to reconstruct any output at all. This holds because a subset of OT executions would output additive shares of the senders' inputs corresponding to 0, while the others would output additive shares corresponding to 1; which is insufficient to reconstruct either of the senders' inputs. We notice that in the above approach, to realize a 4-round protocol, the shares already need to be communicated in the second round. Therefore, the senders' inputs are also required in the second round. To satisfy the delayed-input property, we let each sender S_i follow the steps explained above w.r.t. two uniformly sampled keys k_i^b. Then, S_i sends one-time pad encryptions of their inputs using the sampled keys in the last round. Namely, S_i sends $k_i^b \oplus s_i^b$, $b \in \{0,1\}$ and $i \in [n]$.

We present our protocol formally in Figure 3.1, and summarize the required tools below.

1. A 4-round, delayed-input 1-out-of-2 oblivious transfer protocol $\mathsf{OT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3, \mathsf{OT}_4, \mathsf{OT}_{\mathsf{out}})$ which satisfies the properties of sender simulatability and receiver privacy (Definition 2.5). Here receiver privacy means that a corrupted sender should not be able to tell apart whether the input used by the receiver is 0 or 1.
2. A CPA-secure public-key encryption scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

Figure 3.1: Oblivious Transfer $\mathsf{OT}_{\mathsf{1RnS}}$

Initialization: Each sender S_i (at the end of the third round) uses as its input string vectors $(\{x_{i,l}^0, x_{i,l}^1\}_{l \in [\ell]})$, for $i \in [n]$ (the l-th element $x_{i,l}^b$, is λ-bits long for $b \in \{0,1\}$). The receiver R (at the end of the second round) uses as its input the bit string $m = m_1, \ldots, m_\ell$.

Round 1 **(Receiver).**

1. For each $i \in [n]$, compute $\mathsf{ot}_1^i \leftarrow \mathsf{OT}_1(1^\lambda)$
2. Broadcast $\{\mathsf{ot}_1^i\}_{i \in [n]}$.

Round 2 **(Senders).** Each sender S_i for $i \in [n]$ does the following.

1. Run the setup of the PKE scheme as $(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
2. Compute $\mathsf{ot}_2^i \leftarrow \mathsf{OT}_2(1^\lambda, \mathsf{ot}_1^i)$.
3. Broadcast $\{pk_i\}_{i \in [n]}, \{\mathsf{ot}_2^i\}_{i \in [n]}$.

Round 3 **(Senders).** Each sender S_i for $i \in [n]$ does the following.

1. For each $l \in [\ell]$ sample two λ-bit strings $K_{i,l}^0, K_{i,l}^1 \leftarrow \{0,1\}^\lambda$.
2. For $b \in \{0,1\}$, for $l \in [\ell]$ compute an n-out-of-n additive secret sharing of $K_{i,l}^b$ obtaining shares $\{k_{i,l,j}^b\}_{j \in [n]}$
3. For each $j \in [n]$ and $b \in \{0,1\}$ compute $\{ct_{i,l,j}^b = \mathsf{Enc}(pk_j, k_{i,l,j}^b)\}_{l \in [\ell]}$
4. Broadcast $\{ct_{i,l,j}^0, ct_{i,l,j}^1\}_{j \in [n], l \in [\ell]}$.

Round 3 **(Receiver).** The receiver R computes the following steps.

1. For each $i \in [n]$ compute $\mathsf{ot}_3^i \leftarrow \mathsf{OT}_3(m, (\mathsf{ot}_1^i, \mathsf{ot}_2^i))$.
2. Broadcast $\{\mathsf{ot}_3^i\}_{i \in [n]}$.

Round 4 **(Senders).** Each sender S_i for $i \in [n]$ does the following.

1. For $b \in \{0,1\}$, for $j \in [n]$ compute $\{k_{j,l,i}^b \leftarrow \mathsf{Dec}(sk_i, ct_{j,l,i}^b)\}_{l \in [\ell]}$.
2. For each $l \in [\ell]$ compute $\{X_{i,l}^b = x_{i,l}^b \oplus K_{i,l}^b\}_{b \in \{0,1\}}$.

3. Compute $\mathsf{ot}_4^i \leftarrow \mathsf{OT}_4(\{k^0_{j,l,i}, k^1_{j,l,i}\}_{l\in[\ell],j\in[n]}, (\mathsf{ot}_1^i, \mathsf{ot}_2^i, \mathsf{ot}_3^i))$.
4. Broadcast $\{X^0_{i,l}, X^1_{i,l}\}_{l\in[\ell]}, \{\mathsf{ot}_4^i\}_{i\in[n]}$.

Output Computation.

1. For each $i \in [n]$, R computes $\{k^{m_l}_{i,l,j}\}_{l\in[\ell],j\in[n]} \leftarrow \mathsf{OT}_{\mathsf{out}}(\mathsf{ot}_1^i, \mathsf{ot}_2^i, \mathsf{ot}_3^i, \mathsf{ot}_4^i)$.
2. For each $l \in [\ell]$ and $i \in [n]$ R computes $K^{m_l}_{i,l} = \bigoplus_{j\in[n]} k^{m_l}_{i,l,j}$.
3. For each $l \in [\ell]$ and $i \in [n]$ R outputs $x^{m_l}_{i,l} = X^{m_l}_{i,l} \oplus K^{m_l}_{i,l}$.

Remark 3.2. Note that in Figure 3.1 the senders speak both in the second and the third round. We observe that this communication is only necessary to allow the senders to send each other private messages. In the rest of the paper, we abstract this detail away for a clearer exposition and we assume that the senders have access to private channels, and use them in the second round to send the shares (but we note that this private channel can be implemented using public-key encryption as shown in Figure 3.1). Therefore, in the rest of the paper, when we use $\mathsf{OT}_{\mathsf{1RnS}}$, we will avoid specifying that senders speak also in the third round.

Theorem 3.3. *Assuming that* OT *enjoys the property of sender simulatability and receiver privacy, then* $\mathsf{OT}_{\mathsf{1RnS}}$ *satisfies Definition 3.1. Moreover* $\mathsf{OT}_{\mathsf{1RnS}}$ *makes black-box use of* OT *and* PKE*.*

The correctness follows by inspection. The receiver's privacy follows immediately from the receiver's privacy of OT. We will proceed now through a series of hybrid arguments to prove that $\mathsf{OT}_{\mathsf{1RnS}}$ satisfies also sender privacy, switching from the real-world experiment to the simulated experiment, where the simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ is defined in Figure 3.4. Note that $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ of Figure 3.1 maintains the main-thread during the simulation. Let $\mathcal{A}$ be the adversary that corrupts a subset $\mathcal{I} \subseteq [n]$ of senders and the receiver. Let $\mathcal{H}$ be the set $\{1, \ldots, n\} \setminus \mathcal{I}$.

H_0 The first hybrid H_0 is the experiment where the honest sender $\{S_i\}_{i\in\mathcal{H}}$, interacts with $\mathcal{A}$ following the protocol description with honest inputs $\{\mathbf{x}_i^0, \mathbf{x}_i^1\}_{i\in\mathcal{H}}$. The output of the experiment is the view of the adversary.

H_1 This hybrid works as the previous one except that the ciphertexts sent in the second round between honest parties are encryption of the message 0^λ. More in detail, for $i, j \in \mathcal{H}$ the hybrids computes (in the third round) $\{ct^b_{i,l,j} = \mathsf{Enc}(pk_j, 0^\lambda)\}_{l\in[\ell],b\in\{0,1\}}$. This hybrid is indistinguishable from the previous one due to the CPA security of the encryption scheme PKE.

H_2 This hybrid works as the previous one except that for each $i \in \mathcal{H}$ in the i-th execution of OT the senders' messages are simulated and the input of the receiver m_i^* is extracted. In more detail, the hybrid executes the same steps that $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ performs for the extraction of the receiver's inputs and the simulation of the OT sender messages (which are generated using $\mathsf{OT}.\mathsf{Sim}$). The indistinguishability between this and the previous hybrid follows from the sender simulatability of OT.

H_3 This hybrid works as the previous one except for the way the values $\{k^b_{i,l,i}\}_{l\in[\ell]}$ are generated. In particular, let $\{m^*_i\}_{i\in\mathcal{H}}$ the receiver's input extracted as described in H_2, then to generate the values the hybrid for each $i\in\mathcal{H}$ is performing the following steps:

1. If $|\mathcal{H}|\geq 2$ and there exists $j^*\in\mathcal{H}$ s.t. $m^*_i\neq m^*_j$ for any $j\in\mathcal{H}$, then for each $l\in[\ell]$ pick $\{k^b_{i,l,j}\}_{j\in[n]}$ from the uniform distribution over $\{0,1\}^\lambda$, where $b=m^*_{i,l}$.
2. Otherwise, pick $\{k^{b'}_{i,l,j}\}_{l\in[\ell],j\in[n]\setminus\{i\}}$ and $\{K^{b'}_{i,j}\}_{l\in[\ell],j\in[n]\setminus\{i\},b\in\{0,1\}}$ from the uniform distribution over $\{0,1\}^\lambda$, and set $k^b_{i,l,i}=K^b_{i,l}\bigoplus_{j\in[n]\setminus\{i\}}k^b_{i,l,j}$.
3. For each $l\in[\ell]$ pick $\{k^{1-b}_{i,l,j}\}_{j\in[n]}$ from the uniform distribution over $\{0,1\}^\lambda$, where $b=m^*_{i,l}$.

The perfect indistinguishability between this and the previous hybrid follows from the fact that if condition (1) verifies then the adversary does not have enough shares to reconstruct any of the keys used to encrypt the sender's inputs.

H_5 This hybrid works as the previous one except for the way the values $\{X^{1-b}_{i,l}\}_{l\in[\ell]}$ are generated. In particular, let $\{m^*_i\}_{i\in\mathcal{H}}$ the receiver's input extracted as described in H_2, then to generate the values the hybrid for each $i\in\mathcal{H}$ is performing the following steps:

1. If $|\mathcal{H}|\geq 2$ and there exists $j^*\in\mathcal{H}$ s.t. $m^*_i\neq m^*_j$ for any $j\in\mathcal{H}$, then for each $l\in[\ell]$ pick $\{X^b_{i,l}\}_{l\in[\ell]}$ from the uniform distribution over $\{0,1\}^\lambda$, where $b=m^*_{i,l}$.
2. The values $\{X^{1-b}_{i,l}\}_{l\in[\ell]}$ are sampled from the uniform distribution over $\{0,1\}^\lambda$, where $b=m^*_{i,l}$.

The perfect indistinguishability between this and the previous hybrid follows from the security of the one-time pad.

Figure 3.2: The simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$

Notation: Let $\mathcal{A}$ be the adversary that corrupts a subset $\mathcal{I}\subseteq[n]$ of senders and the receiver. Let $\mathcal{H}$ be the set $\{1,\ldots,n\}\setminus\mathcal{I}$.

Simulation The simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ on input 1^λ computes the following steps:

1. Upon receiving $\{\mathsf{ot}^i_1\}_{i\in[n]}$ from $\mathcal{A}$, for each $i\in\mathcal{H}$ send ot^i_1 to $\mathsf{OT.Sim}$ and collect ot^i_2 from $\mathsf{OT.Sim}$.
2. For each $i\in\mathcal{H}$ run the setup of the PKE scheme as $(pk_i,sk_i)\leftarrow\mathsf{KeyGen}(1^\lambda)$ and broadcast pk_i, ot^i_2.
3. Upon receiving $\{pk_i,\mathsf{ot}^i_2\}_{i\in\mathcal{I}}$ continue with the following steps:
4. For $b\in\{0,1\}$, for $l\in[\ell]$ and $i\in\mathcal{H}$ sample $\{k^b_{i,l,j}\}_{j\in[n]\setminus\{i\}}$ from the uniform distribution over $\{0,1\}^\lambda$.
5. For each $j\in\mathcal{I}$, $i\in\mathcal{H}$ and $b\in\{0,1\}$ compute $\{ct^b_{i,l,j}=\mathsf{Enc}(pk_j,k^b_{i,l,j})\}_{l\in[\ell]}$.
6. For each $j,i\in\mathcal{H}$ and $b\in\{0,1\}$ compute $\{ct^b_{i,l,j}=\mathsf{Enc}(pk_j,0^\lambda)\}_{l\in[\ell]}$.
7. Broadcast $\{ct^0_{i,l,j},ct^1_{i,l,j}\}_{j\in[n],i\in\mathcal{H},l\in[\ell]}$ to $\mathcal{A}$
8. Upon receiving $\{\mathsf{ot}^i_3\}_{i\in[n]}$ from $\mathcal{A}$ for each $i\in\mathcal{H}$ send ot^i_3 to $\mathsf{OT.Sim}$ and proceed to the next stage.

Extraction. The simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ computes the following steps:

1. For each $i_*\in\mathcal{H}$ run $\mathsf{OT.Sim}$ as follows:
   (a) Upon receiving $\overline{\mathsf{ot}}^{i_*}_2$ from $\mathsf{OT.Sim}$ rewind $\mathcal{A}$ in the beginning of the second round and compute the following steps:

i. For each $i \in \mathcal{H} \setminus \{i_*\}$ compute $\mathsf{ot}_2^i \leftarrow \mathsf{OT}_2(1^\lambda, \mathsf{ot}_1^i)$

ii. Set $\mathsf{ot}_2^{i_*} = \overline{\mathsf{ot}}_2^{i_*}$

iii. For each $i \in \mathcal{H}$ run the setup of the PKE scheme as $(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and broadcast pk_i, ot_2^i.

iv. Upon receiving $\{pk_i, \mathsf{ot}_2^i\}_{i \in \mathcal{I}}$ from $\mathcal{A}$ continue with the following steps.

v. For $b \in \{0,1\}$, for $l \in [\ell]$ and $i \in \mathcal{H}$ sample $\{k_{i,l,j}^b\}_{j \in [n] \setminus \{i\}}$ from the uniform distribution over $\{0,1\}^\lambda$.

vi. For each $j \in \mathcal{I}$, $i \in \mathcal{H}$ and $b \in \{0,1\}$ compute $\{ct_{i,l,j}^b = \mathsf{Enc}(pk_j, k_{i,l,j}^b)\}_{l \in [\ell]}$.

vii. For each $j, i \in \mathcal{H}$ and $b \in \{0,1\}$ compute $\{ct_{i,l,j}^b = \mathsf{Enc}(pk_j, 0^\lambda)\}_{l \in [\ell]}$.

viii. Broadcast $\{ct_{i,l,j}^0, ct_{i,l,j}^1\}_{j \in [n], i \in \mathcal{H}, l \in [\ell]}$ to $\mathcal{A}$.

ix. Upon receiving $\{\mathsf{ot}_3^i\}_{i \in [n]}$ from $\mathcal{A}$, set $\overline{\mathsf{ot}}_3^{i_*} = \mathsf{ot}_3^{i_*}$ and send it to $\mathsf{OT.Sim}$.

x. If $\mathsf{OT.Sim}$ outputs $m_{i_*}^*$ go to step (1), otherwise go back to step (a).

Simulation. The simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ sends $\{m_i^*\}_{i \in \mathcal{H}}$ to the ideal functionality obtaining $\{\boldsymbol{\sigma}_i^0, \boldsymbol{\sigma}_i^1\}_{i \in \mathcal{H}}$. Then $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ computes the following steps:

1. For each honest sender S_i for $i \in \mathcal{H}$

   (a) For $b \in \{0,1\}$, for $j \in [n]$ compute $\{k_{j,l,i}^b \leftarrow \mathsf{Dec}(sk_i, ct_{j,l,i}^b)\}_{l \in [\ell]}$.

   (b) For each $l \in [\ell]$ pick $X_{i,l}^{1-b}$ from the uniform distribution over $\{0,1\}^\lambda$, where $b = m_{i,l}^*$.

   (c) If $|\mathcal{H}| \geq 2$ and there exists $j^* \in \mathcal{H}$ s.t. $m_i^* \neq m_j^*$ for any $j \in \mathcal{H}$, then pick $\{k_{i,l,i}^b\}_{l \in [\ell]}$ and $X_{i,l}^b$ from the uniform distribution over $\{0,1\}^\lambda$, where $b = m_{i,l}^*$.

   (d) Otherwise, for each $j \in \mathcal{H}$, for each $l \in [\ell]$, set $b = m_{i,l}^*$ (which is the l-th bit of $m_{i,l}^*$) and compute the following steps:

      i. For each $l \in [\ell]$ sample two λ-bit strings $K_{i,l}^b$ from the uniform distribution over $\{0,1\}^\lambda$ and set $k_{i,l,i}^b = K_{i,l}^b \bigoplus_{j \in [n] \setminus \{i\}} k_{i,l,j}^b$.

      ii. Set $X_{i,l}^b = \sigma_{i,l,j}^b \oplus K_{i,l}^b$.

   (e) For each $i \in \mathcal{H}$ upon receiving the call to the ideal functionality from $\mathsf{OT.Sim}$ acts as the ideal functionality and send $\{k_{i,l,j}^b\}_{j \in [n]}$ in order to get ot_4^i, where $b = m_{i,l}^*$ (which is the l-th bit of $m_{i,l,j}^*$).

2. Broadcast $\{X_{i,l}^0, X_{i,l}^1\}_{i \in \mathcal{H}, l \in [\ell]}, \{\mathsf{ot}_4^i\}_{i \in \mathcal{H}}$ to $\mathcal{A}$.

3. $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ gives in output the view of $\mathcal{A}$ and stop.

Running time of $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$. From the sender simulation security of OT follows that $\mathsf{OT.Sim}$ extracts the input of the receiver from one OT execution in expected polynomial time. Since the simulator executes $\mathsf{OT.Sim}$ a polynomial number of times and each execution is independent of the others the simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ runs in expected polynomial time for the extraction of the receiver's input. Moreover, since $\mathsf{OT.Sim}$ simulates the sender' round of OT in polynomial time (because $\mathsf{OT.Sim}$ maintains the main thread), also $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ runs in polynomial time to simulate the senders' rounds of $\mathsf{OT}_{\mathsf{1RnS}}$.

nRmS *Oblivious Transfer.* This section considers a slight variation of the $\mathsf{1RnS}$ OT introduced in the previous section. The notion of nRmS can be seen as multiple executions (n) of a protocol that realizes the $\mathsf{1RnS}$ notion. That is, we have n receivers, each of them interacting with m distinct senders, and we have the guarantee that each receiver uses the same input when interacting with multiple senders (but different receivers may have different inputs). In more detail our ideal functionality, denoted with $\mathcal{F}_{m,n}$, works as follows: for each receiver R_i, we have m senders $\vec{S}_i = \{S_{1,i}, \ldots, S_{m,i}\}$, with $i \in n$, which engage with the ideal functionality of $\mathcal{F}_{\vec{S}_i,R_i}$ described in Definition 3.1 with a nominated receiver R_i. If Figure 3.3 we propose the formal description of our ideal functionality.

Figure 3.3: $\mathcal{F}_{m,n}$

Functionality $\mathcal{F}_{m,n}$

The functionality interacts with a sets of senders $\{\vec{S}_i\}_{i\in[n]}$ (with $|\vec{S}_i| = m, i \in [n]$) and the receivers $\{R_i\}_{i\in[n]}$, and emulates the behavior of the functionality $\mathcal{F}_{\vec{S}_i,R_i}$ for each $i \in m$ as described below.

- Upon receiving the inputs from all the senders in $\vec{S}_i$ act as $\mathcal{F}_{\vec{S}_i,R_i}$ would act upon receiving the inputs from the senders.
- Upon receiving a message $(\mathsf{receive}, m)$ from R_i act as $\mathcal{F}_{\vec{S}_i,R_i}$ when receiving the input m from the sender $\vec{S}_i$.
- Whenever $\mathcal{F}_{\vec{S}_i,R_i}$ wants to return an output, return out^i_R to R_i.

The security offered from a nRmS OT protocol is essentially the same as the one described in Definition 3.1, with the only difference that more than one receiver can be corrupted.

Definition 3.4. *An* nRmS *OT is defined by the following tuple of algorithms* $\mathsf{OT}_{\vec{S},\vec{R}} := \{(\mathsf{OT}^i_{R,1}, \mathsf{OT}^i_{S,2}, \mathsf{OT}^i_{R,3}, \mathsf{OT}^i_{S,4}, \mathsf{OT}^i_{\mathsf{out}})\}_{i\in n}$*, where, for each* $i \in [n], j \in [m]$ *the sender* $S_{j,i}$ *speaks in the second and fourth rounds using respectively the algorithms* $\mathsf{OT}^i_{S,2}$ *and* $OT^i_{S,4}$ *while for each* $i \in [n]$ *the receiver* R_i *speaks in the first and third rounds using the algorithms* $\mathsf{OT}^i_{R,1}$ *and* $\mathsf{OT}^i_{R,3}$ *respectively. The parties have access to a broadcast channel. We denote with* τ_{OT} *the transcript of the protocol. The algorithm* $\mathsf{OT}^i_{\mathsf{out}}$ *takes as input* τ_{OT}*, the input and randomness of the receiver* R_i *and generates the output* out^i_R *for the receiver* R_i *for each* $i \in [n]$.

Delayed-Input Correctness: *For each* $j \in [n]$*, we denote the input used by the senders* $\vec{S}_j$ *with* $\mathsf{s}_j = \{(\mathsf{s}^0_{i,j}, \mathsf{s}^1_{i,j})\}_{i\in[m]}$ *(i.e., each sender's input, as before, is represented by a pair of vectors of size* ℓ*, where each component is a* λ*-bit string). For every possible input* $\{\mathsf{s}_i, m_i\}_{i\in[n]}$*, where* s_i *denotes the inputs of the senders* $\vec{S}_i$ *and* m_i *the input of the receiver, with* $i \in [n]$ *and where* $|\vec{S}_i| = m$ *we require the following to hold in the case when all the parties are honest:*

$$\Pr\left[\{\mathsf{out}^i_R\}_{i\in[n]} = \mathcal{F}_{m,n}(\{\mathsf{s}_i, m_i\}_{i\in[n]})\right] = 1$$

where out^i_R *denotes the output obtained by the receiver* R_i *at the end of the protocol. We also require the honest senders to receive the input at the end of the third round and the honest receivers to receive it at the end of the second round.*

Sender Privacy: *Let $\mathcal{A}$ be a PPT adversary, with any auxiliary input z, that corrupts an arbitrary subset of senders $S_{i,j}$ with $(i,j) \in \mathcal{I}_S \subseteq [m] \times [n]$ and of receivers R_i with $i \in \mathcal{I}_R \subseteq [n]$. We denote the indices of the honest senders with $\mathcal{H}_S = [m] \times [n] \setminus \mathcal{I}_S$, and the indices of the honest receivers with $\mathcal{H}_R = [n] \setminus \mathcal{I}_R$*

For every set of vector pairs $\{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}_S}$ we require that there exists an expected polynomial time simulator Sim with the following characteristics.

1. *It has black-box access to $\mathcal{A}$.*
2. *It works in two phases. In the first phase, it returns $\{m_k\}_{k\in\mathcal{I}_R}$ (which implicitly denotes the inputs of the corrupted receivers $\{R_k\}_{k\in\mathcal{I}_R}$). In the second phase, the simulator is fed with the vectors $(\{\boldsymbol{\sigma}^0_{i,j}, \boldsymbol{\sigma}^1_{i,j}\}_{(i,j)\in\mathcal{H}_S}$ constructed as follows. For each $(i,j) \in [m] \times [n], d \in \{0,1\}$ $\boldsymbol{\sigma}^d_{i,j} := (\sigma^d_{i,j,1}, \ldots, \sigma^d_{i,j,\ell})$, where for each $(i,j) \in \mathcal{H}_S$ with $j \in \mathcal{I}_R$, $k \in [\ell]$ $\sigma^{m_{j,k}}_{i,j,k} := s^{m_{j,k}}_{i,j,k}, \sigma^{1-m_{j,k}}_{i,j,k} := 0^\lambda$, where $m_{j,k}$ represents the k-th bit of m_j.*
3. *Upon receiving the above input, the simulator returns the output and stops.*

Sender privacy requires that the following two distributions to be computationally indistinguishable

$$\mathrm{View}(\mathcal{A}(z), \mathcal{H}_S, \mathcal{H}_R, \{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}}, \{m_i\}_{i\in\mathcal{H}_R}) \approx \{\mathsf{Sim}^{\mathcal{A}}(1^\lambda, z)\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^\star}$$

where the left part of the above denotes the view of $\mathcal{A}$ during the execution of $\mathsf{OT}_{\vec{S},\vec{R}}$ where the honest senders use the inputs $\{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}}$ and the honest receivers use the inputs $\{m_i\}_{i\in\mathcal{H}_R}$, while $\mathsf{Sim}^{\mathcal{A}}(1^\lambda, z)$ denotes the output of the simulator described above.

Receiver Privacy: *Let $\mathcal{A}$ be a PPT adversary, with auxiliary input $z \in \{0,1\}^\lambda$, that corrupts an arbitrary subset of senders with indices $\mathcal{I}_S \subseteq [m] \times [n]$ and receivers with indices $\mathcal{I}_R \subseteq [n]$ (as before we denote the indices of the honest senders and receivers respectively with $\mathcal{H}_S$ and $\mathcal{H}_R$). For every pairs of receivers' inputs $\{m_i\}_{i\in\mathcal{H}_R}, \{m'_i\}_{i\in\mathcal{H}_R}$ and for every honest senders' inputs $\{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}_S}$ we require that*

$$\mathrm{View}(\mathcal{A}(z), \mathcal{H}_S, \mathcal{H}_R, \{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}}, \{m_i\}_{i\in\mathcal{H}_R}) \approx_c \mathrm{View}(\mathcal{A}(z), \mathcal{H}_S, \mathcal{H}_R, \{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}}, \{m'_i\}_{i\in\mathcal{H}_R})$$

where $\mathrm{View}(\cdot)$ is defined as above.

From $\mathsf{1RnS}$ *Oblivious Transfer to* nRmS *Oblivious Transfer.* We construct an nRmS OT protocol, which we denote with $\mathsf{OT}_{\mathsf{nRmS}}$, satisfying Definition 3.6. The construction of $\mathsf{OT}_{\mathsf{nRmS}}$ is quite simple, as it simply consists of letting each set of senders $\vec{S_i}$ engage in an execution of the $\mathsf{1RnS}$ OT protocol against the receiver R_i for each $i \in n$.

Theorem 3.5. *Assuming that* OT *enjoys the property of sender simulatability and receiver privacy, then* $\mathsf{OT}_{\mathsf{1RnS}}$ *satisfies Definition 3.1. Moreover* $\mathsf{OT}_{\mathsf{1RnS}}$ *makes black-box use of* OT *and* PKE.

The correctness follows by inspection. The receiver's privacy follows immediately from the receiver's privacy of OT. We will proceed now through a series of hybrid arguments to prove that $\mathsf{OT}_{\mathsf{1RnS}}$ satisfies also sender privacy, switching from the real-world experiment to the simulated experiment, where the simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ is defined in Figure 3.4. Note that $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ of Figure 3.1 maintains the main-thread during the simulation. Let $\mathcal{A}$ be the adversary that corrupts a subset $\mathcal{I} \subseteq [n]$ of senders and the receiver. Let $\mathcal{H}$ be the set $\{1, \dots, n\} \setminus \mathcal{I}$.

H_0 The first hybrid H_0 is the experiment where the honest sender $\{S_i\}_{i \in \mathcal{H}}$, interacts with $\mathcal{A}$ following the protocol description with honest inputs $\{\mathbf{x}_i^0, \mathbf{x}_i^1\}_{i \in \mathcal{H}}$. The output of the experiment is the view of the adversary.

H_1 This hybrid works as the previous one except that the ciphertexts sent in the second round between honest parties are encryption of the message 0^λ. More in detail, for $i, j \in \mathcal{H}$ the hybrids computes (in the third round) $\{ct^b_{i,l,j} = \mathsf{Enc}(pk_j, 0^\lambda)\}_{l \in [\ell], b \in \{0,1\}}$. This hybrid is indistinguishable from the previous one due to the CPA security of the encryption scheme PKE.

H_2 This hybrid works as the previous one except that for each $i \in \mathcal{H}$ in the i-th execution of OT the senders' messages are simulated and the input of the receiver m_i^* is extracted. In more detail, the hybrid executes the same steps that $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ performs for the extraction of the receiver's inputs and the simulation of the OT sender messages (which are generated using $\mathsf{OT.Sim}$). The indistinguishability between this and the previous hybrid follows from the sender simulatability of OT.

H_3 This hybrid works as the previous one except for the way the values $\{k^b_{i,l,i}\}_{l \in [\ell]}$ are generated. In particular, let $\{m_i^*\}_{i \in \mathcal{H}}$ the receiver's input extracted as described in H_2, then to generate the values the hybrid for each $i \in \mathcal{H}$ is performing the following steps:

1. If $|\mathcal{H}| \geq 2$ and there exists $j^* \in \mathcal{H}$ s.t. $m_i^* \neq m_j^*$ for any $j \in \mathcal{H}$, then for each $l \in [\ell]$ pick $\{k^b_{i,l,j}\}_{j \in [n]}$ from the uniform distribution over $\{0,1\}^\lambda$, where $b = m^*_{i,l}$.
2. Otherwise, pick $\{k^{b'}_{i,l,j}\}_{l \in [\ell], j \in [n] \setminus \{i\}}$ and $\{K^{b'}_{i,j}\}_{l \in [\ell], j \in [n] \setminus \{i\}, b \in \{0,1\}}$ from the uniform distribution over $\{0,1\}^\lambda$, and set $k^b_{i,l,i} = K^b_{i,l} \bigoplus_{j \in [n] \setminus \{i\}} k^b_{i,l,j}$.
3. For each $l \in [\ell]$ pick $\{k^{1-b}_{i,l,j}\}_{j \in [n]}$ from the uniform distribution over $\{0,1\}^\lambda$, where $b = m^*_{i,l}$.

The perfect indistinguishability between this and the previous hybrid follows from the fact that if condition (1) verifies then the adversary does not have enough shares to reconstruct any of the keys used to encrypt the sender's inputs.

H_5 This hybrid works as the previous one except for the way the values $\{X^{1-b}_{i,l}\}_{l \in [\ell]}$ are generated. In particular, let $\{m_i^*\}_{i \in \mathcal{H}}$ the receiver's input extracted as described in H_2, then to generate the values the hybrid for each $i \in \mathcal{H}$ is performing the following steps:

1. If $|\mathcal{H}| \geq 2$ and there exists $j^* \in \mathcal{H}$ s.t. $m_i^* \neq m_j^*$ for any $j \in \mathcal{H}$, then for each $l \in [\ell]$ pick $\{X^b_{i,l}\}_{l \in [\ell]}$ from the uniform distribution over $\{0,1\}^\lambda$, where $b = m^*_{i,l}$.
2. The values $\{X^{1-b}_{i,l}\}_{l \in [\ell]}$ are sampled from the uniform distribution over $\{0,1\}^\lambda$, where $b = m^*_{i,l}$.

The perfect indistinguishability between this and the previous hybrid follows from the security of the one-time pad.

Figure 3.4: The simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$

Notation: Let $\mathcal{A}$ be the adversary that corrupts a subset $\mathcal{I} \subseteq [n]$ of senders and the receiver. Let $\mathcal{H}$ be the set $\{1, \ldots, n\} \setminus \mathcal{I}$.

Simulation The simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ on input 1^λ computes the following steps:

1. Upon receiving $\{\mathsf{ot}_1^i\}_{i\in[n]}$ from $\mathcal{A}$, for each $i \in \mathcal{H}$ send ot_1^i to $\mathsf{OT.Sim}$ and collect ot_2^i from $\mathsf{OT.Sim}$.
2. For each $i \in \mathcal{H}$ run the setup of the PKE scheme as $(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and broadcast pk_i, ot_2^i.
3. Upon receiving $\{pk_i, \mathsf{ot}_2^i\}_{i\in\mathcal{I}}$ continue with the following steps:
4. For $b \in \{0,1\}$, for $l \in [\ell]$ and $i \in \mathcal{H}$ sample $\{k^b_{i,l,j}\}_{j\in[n]\setminus\{i\}}$ from the uniform distribution over $\{0,1\}^\lambda$.
5. For each $j \in \mathcal{I}$, $i \in \mathcal{H}$ and $b \in \{0,1\}$ compute $\{ct^b_{i,l,j} = \mathsf{Enc}(pk_j, k^b_{i,l,j})\}_{l\in[\ell]}$.
6. For each $j, i \in \mathcal{H}$ and $b \in \{0,1\}$ compute $\{ct^b_{i,l,j} = \mathsf{Enc}(pk_j, 0^\lambda)\}_{l\in[\ell]}$.
7. Broadcast $\{ct^0_{i,l,j}, ct^1_{i,l,j}\}_{j\in[n],i\in\mathcal{H},l\in[\ell]}$ to $\mathcal{A}$
8. Upon receiving $\{\mathsf{ot}_3^i\}_{i\in[n]}$ from $\mathcal{A}$ for each $i \in \mathcal{H}$ send ot_3^i to $\mathsf{OT.Sim}$ and proceed to the next stage.

Extraction. The simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ computes the following steps:

1. For each $i_* \in \mathcal{H}$ run $\mathsf{OT.Sim}$ as follows:
   (a) Upon receiving $\overline{\mathsf{ot}}_2^{i_*}$ from $\mathsf{OT.Sim}$ rewind $\mathcal{A}$ in the beginning of the second round and compute the following steps:
      i. For each $i \in \mathcal{H} \setminus \{i_*\}$ compute $\mathsf{ot}_2^i \leftarrow \mathsf{OT}_2(1^\lambda, \mathsf{ot}_1^i)$
      ii. Set $\mathsf{ot}_2^{i_*} = \overline{\mathsf{ot}}_2^{i_*}$
      iii. For each $i \in \mathcal{H}$ run the setup of the PKE scheme as $(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and broadcast pk_i, ot_2^i.
      iv. Upon receiving $\{pk_i, \mathsf{ot}_2^i\}_{i\in\mathcal{I}}$ from $\mathcal{A}$ continue with the following steps.
      v. For $b \in \{0,1\}$, for $l \in [\ell]$ and $i \in \mathcal{H}$ sample $\{k^b_{i,l,j}\}_{j\in[n]\setminus\{i\}}$ from the uniform distribution over $\{0,1\}^\lambda$.
      vi. For each $j \in \mathcal{I}$, $i \in \mathcal{H}$ and $b \in \{0,1\}$ compute $\{ct^b_{i,l,j} = \mathsf{Enc}(pk_j, k^b_{i,l,j})\}_{l\in[\ell]}$.
      vii. For each $j, i \in \mathcal{H}$ and $b \in \{0,1\}$ compute $\{ct^b_{i,l,j} = \mathsf{Enc}(pk_j, 0^\lambda)\}_{l\in[\ell]}$.
      viii. Broadcast $\{ct^0_{i,l,j}, ct^1_{i,l,j}\}_{j\in[n],i\in\mathcal{H},l\in[\ell]}$ to $\mathcal{A}$.
      ix. Upon receiving $\{\mathsf{ot}_3^i\}_{i\in[n]}$ from $\mathcal{A}$, set $\overline{\mathsf{ot}}_3^{i_*} = \mathsf{ot}_3^{i_*}$ and send it to $\mathsf{OT.Sim}$.
      x. If $\mathsf{OT.Sim}$ outputs $m^*_{i_*}$ go to step (1), otherwise go back to step (a).

Simulation. The simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ sends $\{m_i^*\}_{i\in\mathcal{H}}$ to the ideal functionality obtaining $\{\boldsymbol{\sigma}_i^0, \boldsymbol{\sigma}_i^1\}_{i\in\mathcal{H}}$. Then $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ computes the following steps:

1. For each honest sender S_i for $i \in \mathcal{H}$
   (a) For $b \in \{0,1\}$, for $j \in [n]$ compute $\{k^b_{j,l,i} \leftarrow \mathsf{Dec}(sk_i, ct^b_{j,l,i})\}_{l\in[\ell]}$.
   (b) For each $l \in [\ell]$ pick $X^{1-b}_{i,l}$ from the uniform distribution over $\{0,1\}^\lambda$, where $b = m^*_{i,l}$.
   (c) If $|\mathcal{H}| \geq 2$ and there exists $j^* \in \mathcal{H}$ s.t. $m_i^* \neq m_j^*$ for any $j \in \mathcal{H}$, then pick $\{k^b_{i,l,i}\}_{l\in[\ell]}$ and $X^b_{i,l}$ from the uniform distribution over $\{0,1\}^\lambda$, where $b = m^*_{i,l}$.

(d) Otherwise, for each $j \in \mathcal{H}$, for each $l \in [\ell]$, set $b = m^*_{i,l}$ (which is the l-th bit of $m^*_{i,l}$) and compute the following steps:

   i. For each $l \in [\ell]$ sample two λ-bit strings $K^b_{i,l}$ from the uniform distribution over $\{0,1\}^\lambda$ and set $k^b_{i,l,i} = K^b_{i,l} \bigoplus_{j \in [n] \setminus \{i\}} k^b_{i,l,j}$.

   ii. Set $X^b_{i,l} = \sigma^b_{i,l,j} \oplus K^b_{i,l}$.

(e) For each $i \in \mathcal{H}$ upon receiving the call to the ideal functionality from $\mathsf{OT.Sim}$ acts as the ideal functionality and send $\{k^b_{i,l,j}\}_{j \in [n]}$ in order to get ot^i_4, where $b = m^*_{i,l}$ (which is the l-th bit of $m^*_{i,l,j}$).

2. Broadcast $\{X^0_{i,l}, X^1_{i,l}\}_{i \in \mathcal{H}, l \in [\ell]}, \{\mathsf{ot}^i_4\}_{i \in \mathcal{H}}$ to $\mathcal{A}$.
3. $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ gives in output the view of $\mathcal{A}$ and stop.

Running time of $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$. From the sender simulation security of OT follows that $\mathsf{OT.Sim}$ extracts the input of the receiver from one OT execution in expected polynomial time. Since the simulator executes $\mathsf{OT.Sim}$ a polynomial number of times and each execution is independent of the others the simulator $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ runs in expected polynomial time for the extraction of the receiver's input. Moreover, since $\mathsf{OT.Sim}$ simulates the sender' round of OT in polynomial time (because $\mathsf{OT.Sim}$ maintains the main thread), also $\mathsf{Sim}_{\mathsf{OT}_{\mathsf{1RnS}}}$ runs in polynomial time to simulate the senders' rounds of $\mathsf{OT}_{\mathsf{1RnS}}$.

3.2 Definition of nRmS OT

Definition 3.6. *An* nRmS *OT is defined by the following tuple of algorithms* $\mathsf{OT}_{\vec{S},\vec{R}} := \{(\mathsf{OT}^i_{R,1}, \mathsf{OT}^i_{S,2}, \mathsf{OT}^i_{R,3}, \mathsf{OT}^i_{S,4}, \mathsf{OT}^i_{\mathsf{out}})\}_{i \in n}$*, where, for each* $i \in [n], j \in [m]$ *the sender* $S_{j,i}$ *speaks in the second and fourth rounds using respectively the algorithms* $\mathsf{OT}^i_{S,2}$ *and* $OT^i_{S,4}$ *while for each* $i \in [n]$ *the receiver* R_i *speaks in the first and third rounds using the algorithms* $\mathsf{OT}^i_{R,1}$ *and* $\mathsf{OT}^i_{R,3}$ *respectively. The parties have access to a broadcast channel. We denote with* τ_{OT} *the transcript of the protocol. The algorithm* $\mathsf{OT}^i_{\mathsf{out}}$ *takes as input* τ_{OT}*, the input and randomness of the receiver* R_i *and generates the output* out^i_R *for the receiver* R_i *for each* $i \in [n]$*.*

Delayed-Input Correctness: *For each* $j \in [n]$*, we denote the input used by the senders* $\vec{S}_j$ *with* $\mathsf{s}_j = \{(\mathsf{s}^0_{i,j}, \mathsf{s}^1_{i,j})\}_{i \in [m]}$ *(i.e., each sender's input, as before, is represented by a pair of vectors of size* ℓ*, where each component is a* λ*-bit string). For every possible input* $\{\mathsf{s}_i, m_i\}_{i \in [n]}$*, where* s_i *denotes the inputs of the senders* $\vec{S}_i$ *and* m_i *the input of the receiver, with* $i \in [n]$ *and where* $|\vec{S}_i| = m$ *we require the following to hold in the case when all the parties are honest:*

$$\Pr\left[\{\mathsf{out}^i_R\}_{i \in [n]} = \mathcal{F}_{m,n}(\{\mathsf{s}_i, m_i\}_{i \in [n]})\right] = 1$$

where out^i_R *denotes the output obtained by the receiver* R_i *at the end of the protocol. We also require the honest senders to receive the input at the end of the third round and the honest receivers to receive it at the end of the second round.*

Sender Privacy: *Let $\mathcal{A}$ be a PPT adversary, with any auxiliary input z, that corrupts an arbitrary subset of senders $S_{i,j}$ with $(i,j) \in \mathcal{I}_S \subseteq [m] \times [n]$ and of receivers R_i with $i \in \mathcal{I}_R \subseteq [n]$. We denote the indices of the honest senders with $\mathcal{H}_S = [m] \times [n] \setminus \mathcal{I}_S$, and the indices of the honest receivers with $\mathcal{H}_R = [n] \setminus \mathcal{I}_R$*

For every set of vector pairs $\{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}_S}$ we require that there exists an expected polynomial time simulator Sim with the following characteristics.

1. *It has black-box access to $\mathcal{A}$.*
2. *It works in two phases. In the first phase, it returns $\{m_k\}_{k\in\mathcal{I}_R}$ (which implicitly denotes the inputs of the corrupted receivers $\{R_k\}_{k\in\mathcal{I}_R}$). In the second phase, the simulator is fed with the vectors $(\{\boldsymbol{\sigma}^0_{i,j}, \boldsymbol{\sigma}^1_{i,j}\}_{(i,j)\in\mathcal{H}_S}$ constructed as follows. For each $(i,j) \in [m] \times [n], d \in \{0,1\}$ $\boldsymbol{\sigma}^d_{i,j} := (\sigma^d_{i,j,1}, \ldots, \sigma^d_{i,j,\ell})$, where for each $(i,j) \in \mathcal{H}_S$ with $j \in \mathcal{I}_R$, $k \in [\ell]$ $\sigma^{m_{j,k}}_{i,j,k} := s^{m_{j,k}}_{i,j,k}, \sigma^{1-m_{j,k}}_{i,j,k} := 0^\lambda$, where $m_{j,k}$ represents the k-th bit of m_j.*
3. *Upon receiving the above input, the simulator returns the output and stops.*

Sender privacy requires that the following two distributions to be computationally indistinguishable

$$\mathrm{View}(\mathcal{A}(z), \mathcal{H}_S, \mathcal{H}_R, \{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}}, \{m_i\}_{i\in\mathcal{H}_R}) \approx \{\mathsf{Sim}^{\mathcal{A}}(1^\lambda, z)\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^\star}$$

where the left part of the above denotes the view of $\mathcal{A}$ during the execution of $\mathsf{OT}_{\vec{S},\vec{R}}$ where the honest senders use the inputs $\{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}}$ and the honest receivers use the inputs $\{m_i\}_{i\in\mathcal{H}_R}$, while $\mathsf{Sim}^{\mathcal{A}}(1^\lambda, z)$ denotes the output of the simulator described above.

Receiver Privacy: *Let $\mathcal{A}$ be a PPT adversary, with auxiliary input $z \in \{0,1\}^\lambda$, that corrupts an arbitrary subset of senders with indices $\mathcal{I}_S \subseteq [m] \times [n]$ and receivers with indices $\mathcal{I}_R \subseteq [n]$ (as before we denote the indices of the honest senders and receivers respectively with $\mathcal{H}_S$ and $\mathcal{H}_R$). For every pairs of receivers' inputs $\{m_i\}_{i\in\mathcal{H}_R}, \{m'_i\}_{i\in\mathcal{H}_R}$ and for every honest senders' inputs $\{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}_S}$ we require that*

$$\begin{aligned}\mathrm{View}(\mathcal{A}(z), \mathcal{H}_S, \mathcal{H}_R, \{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}}, \{m_i\}_{i\in\mathcal{H}_R}) \approx_c& \\ \mathrm{View}(\mathcal{A}(z), \mathcal{H}_S, \mathcal{H}_R, &\{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{(i,j)\in\mathcal{H}}, \{m'_i\}_{i\in\mathcal{H}_R})\end{aligned}$$

where $\mathrm{View}(\cdot)$ *is defined as above.*

3.3 Formal Description of $\mathsf{OT}_{\mathsf{nRmS}}$

$\mathsf{OT}_{\mathsf{nRmS}}$ is described in details in Figure 3.5 and makes use of the $\mathsf{1RnS}$ OT protocol $\mathsf{OT}_{\mathsf{1RnS}} = (\mathsf{OT}^1_{\mathsf{1RnS}}, \{\mathsf{OT}^{2,j}_{\mathsf{1RnS}}\}_{j\in[m]}, \mathsf{OT}^3_{\mathsf{1RnS}}, \{\mathsf{OT}^{4,j}_{\mathsf{1RnS}}\}_{j\in[m]}, \mathsf{OT}^{\mathsf{out}}_{\mathsf{1RnS}})$ constructed in Section 3.1.

Figure 3.5: Oblivious Transfer $\mathsf{OT}_{\mathsf{nRmS}}$

Initialization: Each set of m-senders $\vec{S}_i$ (at the end of the third round) has in input string vectors $\{(\mathbf{s}^0_{i,j}, \mathbf{s}^1_{i,j})\}_{j\in[m]}$ (the l-element of the vector $\mathbf{s}^b_{i,j}$ is long λ-bits, $b \in \{0,1\}$,

$l \in [\ell]$). The i-th receiver R_i (at the end of the second round) has input bit string $m^i = m^i_1, \dots, m^i_\ell$, for $i \in [n]$.

Round 1 (Receivers).

1. For each $i \in [n]$ the receiver R_i computes $\mathsf{ot}^{1,i}_{\mathsf{1RnS}} = \mathsf{OT}^1_{\mathsf{1RnS}}(1^\lambda)$ and broadcast $\mathsf{ot}^{1,i}_{\mathsf{1RnS}}$.

Round 2 (Senders).

1. For each $i \in [n]$ each sender S_j in $\vec{S}_i$ computes $\mathsf{ot}^{2,i,j}_{\mathsf{1RnS}} = \mathsf{OT}^{1,j}_{\mathsf{1RnS}}(1^\lambda, \mathsf{ot}^{1,i}_{\mathsf{1RnS}})$ and broadcast $\mathsf{ot}^{2,i,j}_{\mathsf{1RnS}}$.

Round 3 (Receivers).

1. For each $i \in [n]$ the receiver R_i computes $\mathsf{ot}^{3,i}_{\mathsf{1RnS}} = \mathsf{OT}^1_{\mathsf{1RnS}}(m^i, (\mathsf{ot}^{1,i}_{\mathsf{1RnS}}, \{\mathsf{ot}^{2,i,j}_{\mathsf{1RnS}}\}_{j \in [m]}))$ and broadcast $\mathsf{ot}^{3,i}_{\mathsf{1RnS}}$.

Round 4 (Senders).

1. For each $i \in [n]$ each sender S_j in $\vec{S}_i$ computes $\mathsf{ot}^{4,i,j}_{\mathsf{1RnS}} = \mathsf{OT}^{4,j}_{\mathsf{1RnS}}((\mathsf{s}^0_{i,j}, \mathsf{s}^1_{i,j}), (\mathsf{ot}^{1,i}_{\mathsf{1RnS}}, \{\mathsf{ot}^{2,i,j}_{\mathsf{1RnS}}, \}_{j \in [m]}, \mathsf{ot}^{3,i}_{\mathsf{1RnS}})$ and broadcast $\mathsf{ot}^{4,i,j}_{\mathsf{1RnS}}$.

Output Computation.

1. For each receiver R_i computes and outputs $\{s^{m_l}_{i,j}\}_{j \in [m], l \in \ell} = \mathsf{OT}^{\mathsf{out}}_{\mathsf{1RnS}}(\mathsf{ot}^{1,i}_{\mathsf{1RnS}}, \{\mathsf{ot}^{2,i,j}_{\mathsf{1RnS}}, \}_{j \in [m]}, \mathsf{ot}^{3,i}_{\mathsf{1RnS}}, \{\mathsf{ot}^{4,i,j}_{\mathsf{1RnS}}, \}_{j \in [m]})$

Theorem 3.7. *Let $\mathsf{OT}_{\mathsf{1RnS}}$ be the $\mathsf{1RnS}$ oblivious transfer (satisfying Definition 3.1) constructed in Section 3.1, then $\mathsf{OT}_{\mathsf{nRmS}}$ of Figure 3.5 satisfies Definition 3.6. Moreover $\mathsf{OT}_{\mathsf{nRmS}}$ makes black-box use of $\mathsf{OT}_{\mathsf{1RnS}}$.*

Proof. (Sketch) The correctness property follows by inspection and the receiver privacy follows from the receiver privacy of $\mathsf{OT}_{\mathsf{1RnS}}$. The property of sender privacy follows a similar proof to the one given for Theorem 3.5. In particular, the set of hybrids (and the simulation strategy) are the same considered for the proof of Theorem 3.5, the only difference is that the hybrids are iterated for each i-th $\mathsf{OT}_{\mathsf{1RnS}}$ executions, for $i \in [n]$. The transition between the hybrids can be argued similarly to the proof of Theorem 3.5. □

4 Inner Protocol

4.1 Definition of the Inner-Protocol

In this section, we provide the security definition for the inner protocol (which is inspired by [IKSS23]). Then, we show that the protocol described in [PS21] satisfies our security definition. We start the section with an informal description of our inner-protocol definition and its instantiation. Compared to the definition of the inner protocol proposed in [IKSS23], we require some additional properties. Specifically, our inner protocol is formulated with a clear separation between an inputless and an input part. We formalize this separation by splitting the algorithms of the protocol into two parts: one procedure that is responsible for establishing correlated randomness between the parties (and is therefore inputless), denoted by $\Pi^{\mathsf{no\text{-}inp}}$, and one procedure that requires the inputs, denoted by Π^{inp}. Here, we allow the “no-input” procedure to interact with the “input” procedure using an auxiliary input. Furthermore, we also require Π^{inp} to not make any use of computational cryptographic primitives. As mentioned in the introductory section of the paper, this separation is necessary since in our final MPC protocol we will evaluate Π^{inp} inside a garbled circuit and, to guarantee the black-box use of primitives, this procedure cannot perform cryptographic operations. In terms of security, we require two properties:

Equivocality: If the adversary is misbehaving in the first two rounds of $\Pi^{\mathsf{no\text{-}inp}}$, the inputs of the honest parties are leaked to the adversary in the third round.

Semi-maliciousness: If the adversary is computing honestly the first two rounds of $\Pi^{\mathsf{no\text{-}inp}}$, and provide a defense that explains his behavior with respect to the first two messages of $\Pi^{\mathsf{no\text{-}inp}}$ then no information about the honest parties’ inputs is leaked, except what can be inferred from the output of the function.

The difference between our notion and the one proposed in [IKSS23], is that we only require the adversary to behave honestly (and provide a defense) for the first two messages related to $\Pi^{\mathsf{no\text{-}inp}}$. In particular, this means that the simulator does not receive the inputs of the adversary (as the messages of $\Pi^{\mathsf{no\text{-}inp}}$ do not necessitate any input). Hence, the simulator must be able to extract the input of the adversary by just looking at the messages received from the adversary and at the randomness the adversary used to compute the messages of $\Pi^{\mathsf{no\text{-}inp}}$.

After the informal description, we now present the formal definition.

A three-round inner protocol computing a function f is given by a tuple of algorithms $\Pi = (\{\Pi_i^{\mathsf{no\text{-}inp}}\}_{i\in[3]}, \{\Pi_i^{\mathsf{inp}}\}_{i\in 1,2}, \Pi^{\mathsf{out}})$ with the following syntax. The i-th party P_i on private input x_i executes the following steps:

- In the first round compute and broadcast $(\pi_1^i, z_i) \leftarrow \Pi_1^{\mathsf{no\text{-}inp}}(1^\lambda, i)$
- In the second round upon receiving $\{\pi_1^j\}_{j\in[n]\setminus\{i\}}$ compute $(z_i', \pi_2^i) \leftarrow \Pi_2^{\mathsf{no\text{-}inp}}(1^\lambda, i, \{\pi_1^j\}_{j\in[n]\setminus\{i\}}, z_i)$ and $\tilde{\pi}_1^i \leftarrow \Pi_1^{\mathsf{inp}}(i, x_i, z_i')$. Broadcast $(\pi_2^i, \tilde{\pi}_1^i)$.
- In the third round upon receiving $\{\pi_2^j, \tilde{\pi}_1^j\}_{j\in[n]\setminus\{i\}}$ for each $j \in [n] \setminus \{i\}$ compute $(z_i'', \pi_3^i) \leftarrow \Pi_3^{\mathsf{no\text{-}inp}}(1^\lambda, i, z_i', \{\pi_2^j, \pi_1^j\}_{j\in[n]\setminus\{i\}})$ and $\tilde{\pi}_2^i \leftarrow \Pi_2^{\mathsf{inp}}(i, x_i, z_i'', \{\tilde{\pi}_1^j\}_{j\in[n]\setminus\{i\}})$, and broadcast $(\pi_3^i, \tilde{\pi}_2^i)$.
- Upon receiving $\{\pi_3^j, \tilde{\pi}_2^j\}_{j\in[n]\setminus\{i\}}$ for each $j \in [n]\setminus\{i\}$ output $\Pi^{\mathsf{out}}(x_i, \{\pi_3^j, \pi_k^j, \tilde{\pi}_k^j\}_{j\in[n]\setminus\{i\},k\in[2]})$.

Figure 4.1: Security game for the inner protocol

```
$\mathsf{Real}(\mathcal{A}, \{x_i, r_i\}_{i\in H})$

1. For each $i \in H$, compute $(\pi_1^i, z_i) \leftarrow \Pi_1^{\mathsf{no\text{-}inp}}(1^\lambda, i; r_i)$.
2. Send $\{\pi_1^i\}_{i\in H}$ to $\mathcal{A}$.
3. Receive $\{\pi_1^i\}_{i\in M}$ from $\mathcal{A}$.
4. For each $i \in H$, compute $(z_i', \pi_2^i; r_i) := \Pi_2^{\mathsf{no\text{-}inp}}(1^\lambda, i, \{\pi_1^j\}_{j\in[n]\setminus\{i\}}, z_i)$ and $\tilde{\pi}_1^i := \Pi_1^{\mathsf{inp}}(i, x_i, z_i'; r_i)$. Broadcast $(\pi_2^i, \tilde{\pi}_1^i)$.
5. Send $\{\pi_2^i, \tilde{\pi}_1^i\}_{i\in H}$ to $\mathcal{A}$.
6. Receive $\{\pi_2^i, \tilde{\pi}_1^i, r_i\}_{i\in M}$ from $\mathcal{A}$.
7. For each $i \in H$, compute $(z_i'', \pi_3^i) \leftarrow \Pi_3^{\mathsf{no\text{-}inp}}(1^\lambda, i, z_i', \{\pi_2^j, \pi_1^j\}_{j\in[n]\setminus\{i\}})$ and $\tilde{\pi}_2^i \leftarrow \Pi_2^{\mathsf{inp}}(i, x_i, z_i'', \{\tilde{\pi}_1^j\}_{j\in[n]\setminus\{i\}})$
8. Send $\{\pi_3^i, \tilde{\pi}_2^i\}_{i\in H}$ to $\mathcal{A}$.
9. Receive $\{\pi_3^i, \tilde{\pi}_2^i\}_{i\in M}$ from $\mathcal{A}$.
10. Output the view of $\mathcal{A}$ and $\{\Pi^{\mathsf{out}}(x_i, \{\pi_3^j, \pi_k^j, \tilde{\pi}_k^j\}_{j\in[n]\setminus\{i\},k\in[2]})\}_{i\in H}$
```

```
$\mathsf{Ideal}(\mathcal{A}, \mathsf{Sim}_\Pi, \{x_i\}_{i\in H})$

1. For each $i \in H$, compute $\pi_1^i \leftarrow \mathsf{Sim}_\Pi(1^\lambda, i)$.
2. Send $\{\pi_1^i\}_{i\in H}$ to $\mathcal{A}$.
3. Receive $\{\pi_1^i\}_{i\in M}$ from $\mathcal{A}$.
4. For each $i \in H$, compute $(\pi_2^i, \tilde{\pi}_1^i) \leftarrow \mathsf{Sim}_\Pi(\{\pi_1^i\}_{i\in M})$
5. Send $\{\pi_2^i, \tilde{\pi}_1^i\}_{i\in H}$ to $\mathcal{A}$.
6. Receive $\{\pi_2^i, \tilde{\pi}_1^i, r_i\}_{i\in M}$ from $\mathcal{A}$.
7. Check if the messages sent by the corrupted parties in $\{\pi_1^i, \pi_2^i\}_{i\in M}$ are consistent with $\{r_i\}_{i\in M}$.
8. **Semi-Malicious Security:** if they are consistent:
   (a) $\{x_i\}_{i\in M} \leftarrow \mathsf{Sim}_\Pi(1^\lambda, i, \{\pi_2^i, \tilde{\pi}_1^i\}_{i\in M}, \{r_i\}_{i\in M})$
   (b) For each $i \in H$, compute $(\pi_3^i, \tilde{\pi}_2^i) \leftarrow \mathsf{Sim}_\Pi(i, f(x_1, \ldots, x_n))$.
9. **Equivocality:** if they are not consistent:
   (a) For each $i \in H$, compute $(\pi_3^i, \tilde{\pi}_2^i) \leftarrow \mathsf{Sim}_\Pi(1^\lambda, i, \{x_i\}_{i\in H})$
10. Send $\{\pi_3^i, \tilde{\pi}_2^i\}_{i\in H}$ to $\mathcal{A}$.
11. Receive $\{\pi_3^i, \tilde{\pi}_2^i\}_{i\in M}$ from $\mathcal{A}$.
12. Output the view of $\mathcal{A}$ and $\{\Pi^{\mathsf{out}}(x_i, \{\pi_3^j, \pi_k^j, \tilde{\pi}_k^j\}_{j\in[n]\setminus\{i\},k\in[2]})\}_{i\in H}$.
```

Definition 4.1. *As inner protocol Π for f is secure if it satisfies the following properties:*
Correctness: *The protocol Π correctly computes a function f if for every choice of inputs x_i for party P_i, $\Pr[\Pi^{\mathsf{out}}(\tau_\Pi)) = f(x_1, \ldots, x_n)] = 1$, where τ_Π denotes the transcript of the protocol Π when the input of P_i is x_i.*

Security: *Let $\mathcal{A}$ be an adversary corrupting a subset of the parties indexed by the set M and let H be the set of indices denoting the honest parties. We require the existence of a simulator Sim_Π such that for any choice of honest parties inputs $\{x_i\}_{i\in H}$, the real world* Real$(\mathcal{A}, \{x_i, r_i\}_{i\in H})$ *and the ideal world* Ideal$(\mathcal{A}, \mathsf{Sim}_\Pi, \{x_i\}_{i\in H})$ *are indistinguishable. Here, the real and ideal experiments are described in Fig. 4.1 and for each $i \in H$, r_i is uniformly chosen.*

Primitive Independency: *We require that the operations performed by the algorithm Π_2^{inp} are independent of any computational cryptographic primitive.*

After presenting the formal definition of the inner-protocol, we describe how this definition can be realized.

4.2 Realization of the Inner-Protocol

The work of [PS21] realizes a 3-round MPC protocol $\mathit{\Pi}_{\mathsf{PS}}$ for any functionality making black-box use of oblivious transfer against an active adversary in the CRS model. To realize our inner protocol, as in the work of Ishai et al. [IKSS23], we rely on $\mathit{\Pi}_{\mathsf{PS}}$ which, in turn, uses the round-collapsing compiler of [GS18, GIS18]. We now proceed with a brief overview of [GS18, GIS18] and [PS21] and explain why it satisfies the definition described above. In [GS18] the authors propose a compiler $\mathit{\Pi}_{\mathsf{GS}}$ that collapses an MPC protocol $\mathit{\Phi}$ with an arbitrary number of rounds into a two-round protocol. The main technical contribution of [GS18] is to replace the interactions between the parties with a mechanism that emulates these interactions via nested layers of garbled circuits. The interaction between the garbled circuits is then performed using an oblivious transfer. Subsequently, in [GIS18], the authors show that the above compiler can make black-box use of the underlying primitives if $\mathit{\Phi}$ is an information-theoretic protocol in the OT correlations model. In [PS21], the authors show how to execute $\mathit{\Pi}_{\mathsf{GS}}$ and generate the corresponding OT correlations in three rounds. In particular, they present a 3-round protocol $\mathit{\Gamma}$ which generates the OT correlations and provide them as input to the first layer of the garbled circuits of $\mathit{\Pi}_{\mathsf{GS}}$. To be more precise, in the first round of $\mathit{\Pi}_{\mathsf{PS}}$ the parties compute the first round of $\mathit{\Gamma}$, then they execute the subsequent rounds of $\mathit{\Pi}_{\mathsf{GS}}$ and $\mathit{\Gamma}$ in parallel, using the inputs of $\mathit{\Gamma}$ as an additional input to $\mathit{\Pi}_{\mathsf{GS}}$. Note that $\mathit{\Gamma}$ is generating a preprocessing phase (i.e. OT correlations) for $\mathit{\Pi}_{\mathsf{GS}}$ and therefore does not need the inputs of the parties. Our only modification to the above-described protocol is syntactical, as we divide the protocol into two sub-protocols $\Pi^{\mathsf{no\text{-}inp}}$ and Π^{inp}. We observe that the generation of the leveled garbled circuits (and corresponding labels) executed in $\mathit{\Pi}_{\mathsf{GS}}$ can be seen as a preprocessing phase, while the parties' inputs are only involved in the process of selecting the appropriate labels for the layers of the garbled circuits. Therefore, $\Pi^{\mathsf{no\text{-}inp}}$, besides executing the protocol Γ, is also used to setup the garbled circuits and OTs that are part of the computation of $\mathit{\Pi}_{\mathsf{GS}}$. The labels of the garbled circuits generated using $\Pi^{\mathsf{no\text{-}inp}}$ are then passed as auxiliary information to Π^{inp}. Finally, Π^{inp} outputs the correct set of labels based on the parties' actual inputs.

Before formally presenting the inner protocol, we need to introduce, in the subsequent sections, some preliminary tools which are used to construct the final protocol.

4.3 Oblivious Transfer with Equivocal Receiver Security

Syntax. Let $\mathsf{OT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{out}_{\mathsf{OT}})$ be a two-round oblivious transfer protocol. The OT_1 algorithm takes in a unary representation of the security parameter 1^λ and the receiver's choice bit b and outputs the first round message ot_1 along with a secret key sk. The OT_2 algorithm takes in the first round message ot_1, the sender inputs m_0, m_1 and outputs the sender message ot_2. The $\mathsf{out}_{\mathsf{OT}}$ algorithm takes in the sender message ot_2 and the secret key sk and outputs the message m_b.

Definition 4.2 ([IKSS23]). *We say that the* OT *protocol is a two-round oblivious transfer with equivocal receiver security [GS18, PS21] if it satisfies the following properties:*

Correctness: *For every input* b *of the receiver and* m_0, m_1 *of the sender:*

$$\Pr[\mathsf{out}_{\mathsf{OT}}(\mathsf{ot}_2, (b, \mathsf{sk})) = m_b] = 1,$$

where $(\mathsf{ot}_1, \mathsf{sk}) \leftarrow \mathsf{OT}_1(1^\lambda, b)$ *and* $\mathsf{ot}_2 \leftarrow \mathsf{OT}_2(\mathsf{ot}_1, m_0, m_1)$.

Equivocal Receiver Security: *There exists a special algorithm* $\mathsf{Sim}^{Eq}_{\mathsf{OT}}$ *that on input* 1^λ *outputs* $(\mathsf{ot}_1, \mathsf{sk}_0, \mathsf{sk}_1)$ *such that for any* $b \in \{0,1\}$,

$$\{(\mathsf{ot}_1, \mathsf{sk}_b) : (\mathsf{ot}_1, \mathsf{sk}_0, \mathsf{sk}_1) \leftarrow \mathsf{Sim}^{Eq}_{\mathsf{OT}}(1^\lambda)\} \approx_c \{(\mathsf{ot}_1, \mathsf{sk}) : (\mathsf{ot}_1, \mathsf{sk}) \leftarrow \mathsf{OT}_1(1^\lambda, b)\}$$

Sender Privacy: *For any input* m_0, m_1 *of the sender and any bit* b *and a string* $r \in \{0,1\}^*$*:*

$$\{b, r, \mathsf{ot}_1 \coloneqq \mathsf{OT}_1(1^\lambda, b; r), \mathsf{OT}_2(\mathsf{ot}_1, m_0, m_1)\} \approx_c$$
$$\{b, r, \mathsf{ot}_1 \coloneqq \mathsf{OT}_1(1^\lambda, b; r), \mathsf{OT}_2(\mathsf{ot}_1, m_b, m_b)\}$$

A protocol that achieves the described security requirements is the two-round OT protocol with equivocal receiver security of [GS18,IKSS21]. In [IKSS23, Theorem 3.4] it is specified that such a protocol can be constructed from the black-box use of a two-message semi-malicious OT protocol (Definition 2.1).

4.4 Protocol for the Double Selection Functionality DS [PS21]

$\mathbf{\Gamma}$ is a three-round protocol that computes the double selection functionality DS [PS21] with publicly decodable transcript. By public decodable, we mean that output can be computed without having access to the private randomness of the parties. The double selection functionality is a multiparty functionality where P_1 has input $(\alpha, r) \in \{0,1\} \times \{0,1\}$, P_2 has input $(s_0, s_1) \in \{0,1\} \times \{0,1\}$ and for each $i \in [n]$, P_i has additional inputs (z_0^i, z_1^i). The output of the functionality is given by $(s_\alpha \oplus r, \{z^i_{s_{\alpha \oplus r}}\}_{i \in [n]})$. The protocol $\mathbf{\Gamma}$ has the following syntax:

Syntax. The three-round protocol $\mathbf{\Gamma}$ computing the double selection function DS is given by a tuple of algorithsm $(\mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \mathbf{\Gamma}_3, \mathsf{out}_{\mathbf{\Gamma}})$ with the following syntax. For each round $r \in [3]$, the i'th party in the protocol runs $\mathbf{\Gamma}_r$ on 1^λ, the index i, the private input x_i and the transcript of the protocol for the first $(r-1)$ rounds to obtain $\gamma_{r,i}$. It sends $\gamma_{r,i}$ to every other party via a broadcast channel. At the end of the interaction, the public output procedure $\mathsf{out}_{\mathbf{\Gamma}}(\{\gamma_{i,r}\}_{\in[n], r\in[3]})$ is executed to compute the final output.

The security definition that we require from the protocol $\mathbf{\Gamma}$ is the same as in [IKSS23, Section 8.2.1] and is defined as follows:

Definition 4.3. *The protocol* $\mathbf{\Gamma}$ *is said to compute the double selection functionality* DS *if it satisfies the following properties:*

Correctness: *The protocol Γ correctly computes the double selection function* DS *if for every choice of inputs x_i for party P_i,*

$$\Pr[\mathsf{out}_\Gamma(\{\gamma_{i,r}\}_{\in[n],r\in[3]}) = \mathsf{DS}(x_1, \ldots, x_n)] = 1$$

where $\{\gamma_{i,r}\}_{\in[n],r\in[3]}$ denotes the transcript of the protocol Γ when the input of P_i is x_i.

Security: *Let x_i be the input used by party P_i in the protocol Γ. Let $\mathcal{A}$ be any malicious adversary that is corrupting a set of parties indexed by M and let H be the set of honest parties. We require the existence of a stateful simulator* Sim_Γ *such that:* Sim_Γ*, on input 1^λ and i, outputs the first round message $\gamma_{1,i}$ and* td_i*. Using* td_i *and the input x_i of party P_i, there is a polynomial time algorithm* Equiv *that outputs the second and third round messages of the protocol Γ such that the view of any adversary in the real execution with the honest parties is computationally indistinguishable to the distribution of messages generated as above.*

$$\mathrm{Real}_\Gamma(\mathcal{A}, \{x_i, r_i\}_{i\in H}) \approx_c \mathrm{Ideal}_\Gamma(\mathcal{A}, \mathsf{Sim}_\Gamma, \{x_i\}_{i\in H}),$$

where the real and ideal experiments are described in Figure 4.2 and for each $i \in H$, r_i is uniformly chosen.

Figure 4.2: Security game for the protocol $\mathbf{\Gamma}$

$\mathsf{Real}_{\Gamma}(\mathcal{A}, \{x_i, r_i\}_{i \in H})$

1. For each $i \in H$, compute $\gamma_1^i := \mathbf{\Gamma}_1(1^\lambda, i, x_i; r_i)$
2. Send $\{\gamma_{1,i}\}_{i \in H}$ to $\mathcal{A}$.
3. Receive $\{\gamma_{1,i}, (x_i, r_i)\}_{i \in M}$ from $\mathcal{A}$.
4. For round $r \in \{2, 3\}$:
   - (a) For each $i \in H$, compute $\gamma_{r,i} := \mathbf{\Gamma}_r(1^\lambda, i, x_i, \{\gamma_{(r-1),i}\}_{i \in [n]}; r_i)$.
   - (b) Send $\{\gamma_{r,i}\}_{i \in H}$ to $\mathcal{A}$.
   - (c) Receiver $\{\gamma_{r,i}\}$ from $\mathcal{A}$.
5. Output the view of $\mathcal{A}$ and $\mathsf{out}_{\Gamma}(\{\gamma_{i,r}\}_{\in [n], r \in [3]})$.

$\mathsf{Ideal}_{\Gamma}(\mathcal{A}, \mathsf{Sim}_{\Gamma}, \{x_i\}_{i \in H})$

1. For each $i \in H$, compute $\gamma_{1,i} := \mathsf{Sim}_{\Gamma}(1^\lambda, i)$.
2. Send $\{\gamma_{1,i}\}_{i \in H}$ to $\mathcal{A}$.
3. Receive $\{\gamma_{1,i}, (x_i, r_i)\}_{i \in M}$ from $\mathcal{A}$.
4. For round $r \in \{2, 3\}$:
   - (a) Check if the messages sent by the corrupted parties in $\{\gamma_{(r-1),i}\}_{i \in [n]}$ are consistent with $\{r_i\}_{i \in M}$.
   - (b) **Semi-Malicious Security:** if they are consistent:
     - i. For each $i \in H$, compute $\gamma_r^i := \mathsf{Sim}_{\Gamma}(1^\lambda, i, \{x_i, r_i\}_{i \in M}, \mathsf{DS}(x_1, \ldots, x_n), \{\gamma_{(r-1),i}\}_{i \in [n]})$.[a]
   - (c) **Equivocality:** if they are not consistent:
     - i. For each $i \in H$, compute $\gamma_r^i := \mathsf{Sim}_{\Gamma}(1^\lambda, i, \{x_i\}_{i \in H}, \{\gamma_{(r-1),i}\}_{i \in [n]})$.
   - (d) Send $\{\gamma_{r,i}\}_{i \in H}$ to $\mathcal{A}$.
   - (e) Receiver $\{\gamma_{r,i}\}$ from $\mathcal{A}$.
5. Output the view of $\mathcal{A}$ and $\mathsf{out}_{\Gamma}(\{\gamma_{i,r}\}_{\in [n], r \in [3]})$.

[a] Here, the value $\mathsf{DS}(x_1, \ldots, x_n)$ is the reply of the ideal functionality on the query $\{x_i, r_i\}_{i \in M}$.

A protocol that realizes this notion is the protocol presented in [IKSS23, Figure 10] which, in turn, adapts the work of [PS21] and can be instantiated using black-box use of the OT with equivocal receiver security.

4.5 Conforming Protocols

Now, we introduce the notion of conforming protocols. This part is taken from [PS21].

Consider an n-party deterministic[10] 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

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

$\mathsf{pre}, \mathsf{post}$, and computations steps or what we call *actions* $\phi_1, \ldots, \phi_T$. The protocol Φ proceeds in three stages: the pre-processing stage, the computation stage and the output stage.

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

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

where $v_i[(i-1)\ell/n + 1, (i-1)\ell/n + m]$ denotes the bits of the string v_i in the positions $[(i-1)\ell/n + 1, (i-1)\ell/n + m]$. P_i retains v_i as the secret information and broadcasts z_i to every other party. We require that $v_i[k] = 0$ for all $k \in [\ell] \setminus \{(i-1)\ell/n + 1, \ldots, i\ell/n\}$.[11].

Computation phase: For each $i \in [n]$, party P_i sets $\mathsf{st} := (z_1 \| \ldots \| z_n)$. Next, for each $t \in [T]$ 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 $\mathsf{st}[h] = \mathsf{NAND}\left(\mathsf{st}[f] \oplus v_i[f], \mathsf{st}[g] \oplus v_i[g]\right) \oplus v_i[h]$ and broadcasts $\mathsf{st}[h]$ to every other party.
3. Every party P_j for $j \neq i$ updates $\mathsf{st}[h]$ to the bit value received from P_i.

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

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

In [GS18, Section 4.2] it has been shown how any MPC protocol can be turned into a conforming protocol. This does not change anything w.r.t. the way in which the underlying primitives are being invoked. In our work, we use the perfect/statistical protocol in the OT correlations model, e.g., [Kil88, IPS08].

4.6 Formal Description of the Three Round Inner Protocol

In this section, we recall the protocol described in [PS21] casting it using the syntax of our inner-protocol defined in Section 4.1. Further, we prove that it satisfies Definition 4.1.

For the construction, we require all the primitives introduced in this section so far. Namely, the protocol Γ (Section 4.4) for the double selection functionality, the OT protocol OT (Section 4.3) with equivocal receiver security, as well as a conforming protocol Φ (Section 4.5).

In the protocol below, we denote by κ the number of required random OT correlations between party P_i (acting as the receiver) and P_j (acting as the sender) in the protocol Φ. Furthermore, we require a helper function $\mathsf{GetIndex}$:

$\mathsf{GetIndex}(i, j, k)$**:** Takes i, j, k as inputs (where $i, j \in [n], i \neq j$ and $k \in [\kappa]$) and returns an index 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).

[11] Here, as in [PS21], we slightly differ from the formulation used in [GS18, GIS18]. In their work, pre is defined to additionally take x_i as an 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 $v_i[(i-1)\ell/n + 1, (i-1)\ell/n + m]$.

Figure 4.3: Inner-Protocol $\mathit{\Pi}$ for function f

Round 1.

$\Pi_1^{\mathsf{no\text{-}inp}}(1^\lambda, i; r_i)$**:**

1. Sample v_i randomly.
2. Execute the protocol $\boldsymbol{\Gamma}$ $(n-1)^2\kappa$-times. In those k-th executions (for $k \in [\kappa]$), we denote the message $\gamma_{1,i}^{j,l,k}$ as the message generated in the setting where P_j behaves as party P_1 and P_l behaves as party P_2, sent by party i. The messages in those executions are generated as follows:
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma_{1,i}^{i,j,k} \leftarrow \boldsymbol{\Gamma}_1(1^\lambda, 1, (\alpha_k^{i,j}, r_k^{i,j}))$, on behalf of P_1, using $\alpha_k^{i,j}$, a uniformly chosen bit, $r_k^{i,j} := v_i[\mathsf{GetIndex}(i,j,k)]$ and $(\mathsf{sk}_{k,0}^{i,i,j}, \mathsf{sk}_{k,1}^{i,i,j})$ are generated using $\mathsf{Gen}(1^\lambda)$.
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma_{1,i}^{j,i,k} \leftarrow \boldsymbol{\Gamma}_1(1^\lambda, 2, (y_{k,0}^{j,i}, y_{k,1}^{j,i}))$, on behalf of P_2, using uniformly chosen bits $y_{k,0}^{j,i}, y_{k,1}^{j,i}$ and $(\mathsf{sk}_{k,0}^{i,j,i}, \mathsf{sk}_{k,1}^{i,j,i})$ are generated using $\mathsf{Gen}(1^\lambda)$.
   - For all $j, l \in [n]$ with $l \neq j$, generate $\gamma_{1,i}^{j,l,k} \leftarrow \boldsymbol{\Gamma}_1(1^\lambda, i, (\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l}))$ with $\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l}$ being generated using $\mathsf{Gen}(1^\lambda)$.[a]
3. Let $\mathsf{st}_\Gamma := (v_i, \{\alpha_k^{i,j}, y_{k,0}^{j,i}, y_{k,1}^{j,i}\}_{j\in[n]\setminus\{i\}, k\in[\kappa]})$.
4. Output $\{\{\gamma_{1,i}^{i,j,k}, \gamma_{1,i}^{j,i,k}\}_{j\in[n]\setminus\{i\}}, \{\gamma_{1,i}^{j,l,k}\}_{j,l\in[n], l\neq j}\}_{k\in[\kappa]}$.

Round 2.

$\Pi_2^{\mathsf{no\text{-}inp}}(1^\lambda, i; r_i)$**:**

1. 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: $(\mathsf{ot}_1^{i,t,\alpha,\beta}, \mu^{i,t,\alpha,\beta}) \leftarrow \mathsf{OT}_1(v_i[h] \oplus \mathsf{NAND}(v_i[f] \oplus \alpha, v_i[g] \oplus \beta))$.
2. Continue the execution of the $(n-1)^2\kappa$ instances of the protocol $\boldsymbol{\Gamma}$. The messages in those executions are generated as follows:
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma_{2,i}^{i,j,k} \leftarrow \boldsymbol{\Gamma}_2(1^\lambda, 1, (\alpha_k^{i,j}, r_k^{i,j}))$, on behalf of P_1.
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma_{2,i}^{j,i,k} \leftarrow \boldsymbol{\Gamma}_2(1^\lambda, 2, (y_{k,0}^{j,i}, y_{k,1}^{j,i}))$, on behalf of P_2.
   - For all $j, l \in [n]$ with $l \neq j$, generate $\gamma_{2,i}^{j,l,k} \leftarrow \boldsymbol{\Gamma}_2(1^\lambda, i, (\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l}))$.
3. Output $\{\mathsf{ot}_1^{i,t,\alpha,\beta}\}_{t\in A_i, \alpha,\beta\in\{0,1\}}$, $\{\{\gamma_{2,i}^{i,j,k}, \gamma_{2,i}^{j,i,k}\}_{j\in[n]\setminus\{i\}}, \{\gamma_{2,i}^{j,l,k}\}_{j,l\in[n], l\neq j}\}_{k\in[\kappa]}$.

$\Pi_1^{\mathsf{inp}}(1^\lambda, i, x_i, \mathsf{st}_\Gamma; \tilde{r}_i)$**:**

1. Set $x_i^{\mathsf{part}} := (x_i, \{\alpha_k^{i,j}, y_{k,0}^{j,i}, y_{k,1}^{j,i}\}_{j\in[n]\setminus\{i\}, k\in[\kappa]})$.
2. Set $z_i^{\mathsf{part}} := x_i^{\mathsf{part}} \oplus v_i[(i-1)\ell/n + (n-1)\kappa + 1, i\ell/n]$.
3. Output z_i^{part}.

Round 3.

$\Pi_3^{\mathsf{no\text{-}inp}}(1^\lambda, i; r_i)$**:**

1. Set $\mathsf{lab}^{i,T+1} := \{\mathsf{lab}_{k,0}^{i,T+1}, \mathsf{lab}_{k,1}^{i,T+1}\}_{k\in[\ell]}$ where for each $k \in [\ell]$ and $b \in \{0,1\}, \mathsf{lab}^{i,T+1} := \perp$.
2. For each t from T down to 1, do the following:

(a) Let ϕ_t as (i^*, f, g, h).

(b) If $i = i^*$, compute $(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \{\mu^{i,t,\alpha,\beta}\}_{\alpha,\beta}, \perp, \mathsf{lab}^{i,t+1}])$.

(c) If $i \neq i^*$, then for every $\alpha, \beta \in \{0,1\}$, set $\mathsf{ot}_2^{i^*,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{ot}_1^{i^*,t,\alpha,\beta}, \mathsf{lab}_{h,0}^{i,t+1}, \mathsf{lab}_{h,1}^{i,t+1})$ and compute $(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \perp, \{\mathsf{ot}_2^{i,t,\alpha,\beta}\}_{\alpha,\beta}, \mathsf{lab}^{i,t+1}])$.

3. For each $j, j' \in [n]$ with $j \neq j'$ and $k \in [\kappa]$, compute $\mathsf{ct}_{k,0}^{i,j,j'} = \mathsf{Enc}(\mathsf{sk}_{k,0}^{i,j,j'}, \mathsf{lab}_{\mathsf{GetIndex}(j,j',k),0}^{i,1})$ and $\mathsf{ct}_{k,1}^{i,j,j'} = \mathsf{Enc}(\mathsf{sk}_{k,1}^{i,j,j'}, \mathsf{lab}_{\mathsf{GetIndex}(j,j',k),1}^{i,1})$.
4. Set $\mathsf{st}_{\mathsf{lab}} := \mathsf{lab}^{i,1}$.
5. Continue the execution of the $(n-1)^2\kappa$ instances of the protocol $\mathbf{\Gamma}$. The messages in those executions are generated as follows:
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma_{3,i}^{i,j,k} \leftarrow \mathbf{\Gamma}_3(1^\lambda, 1, (\alpha_k^{i,j}, r_k^{i,j}))$, on behalf of P_1.
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma_{3,i}^{j,i,k} \leftarrow \mathbf{\Gamma}_3(1^\lambda, 2, (y_{k,0}^{j,i}, y_{k,1}^{j,i}))$, on behalf of P_2.
   - For all $j, l \in [n]$ with $l \neq j$, generate $\gamma_{3,i}^{j,l,k} \leftarrow \mathbf{\Gamma}_3(1^\lambda, i, (\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l}))$.
6. Output $\{\widetilde{C}^{i,t}\}_{t\in[T]}$, $(\mathsf{ct}_{k,0}^{i,j,j'}, \mathsf{ct}_{k,1}^{i,j,j'})_{j,j'\in[n],j\neq j',k\in[\kappa]}$ and $\{\{\gamma_{3,i}^{i,j,k}, \gamma_{3,i}^{j,i,k}\}_{j\in[n]\setminus\{i\}}, \{\gamma_{3,i}^{j,l,k}\}_{j,l\in[n],l\neq j}\}_{k\in[\kappa]}$.

$\Pi_2^{\mathsf{inp}}(1^\lambda, i, x_i, \mathsf{st}_{\mathsf{lab}}; \tilde{r}_i)$**:**

1. Set $\mathsf{st} := \Big((0^{(n-1)\kappa} \| z_1^{\mathsf{part}}) \| \ldots \| (0^{(n-1)\kappa} \| z_n^{\mathsf{part}})\Big)$.
2. Output $\{\mathsf{lab}_{k,\mathsf{st}[k]}^{i,1}\}_{k\notin[(j-1)\ell/n+1,(j-1)\ell/n+(n-1)\kappa]}$.

Output Computation.

1. For each $j, j' \in [n]$ such that $j \neq j'$ and for each $k \in [\kappa]$, let $\eta := \mathsf{GetIndex}(i, j, k)$, do the following:

   (a) Compute the output of the protocol $\mathbf{\Gamma}$, i.e., $(z_\eta, \{\mathsf{sk}^{s,j,j'}\}_{s\in[n]}) := \mathsf{out}_\Gamma(\{\gamma_{l,r}^{j,j',k}\}_{l\in[n],r\in[3]})$.

   (b) Reset $\mathsf{st}[\eta] = z_\eta$.

   (c) For each $s \in [n]$, set $\mathsf{lab}_{\eta,\mathsf{st}[\eta]}^{s,1} := \mathsf{Dec}(\mathsf{sk}_{k,z_\eta}^{s,j,j'}, \mathsf{ct}_{k,\mathsf{st}[\eta]}^{s,j,j'})$.
2. 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 third-round message.
3. For each t from 1 to T do:

   (a) Parse ϕ_t as (i^*, f, g, h).

   (b) Compute $((\alpha, \beta, \gamma), \mu, \widetilde{\mathsf{lab}}^{i^*,t+1}) := \mathsf{Eval}(\widetilde{C}^{i^*,t}, \widetilde{\mathsf{lab}}^{i^*,t})$.

   (c) Set $\mathsf{st}[h] := \gamma$.

   (d) For each $j \neq i^*$, do:

      i. Compute ot_2,

      ii. Recover $\mathsf{lab}_{h,\mathsf{st}[h]}^{j,t+1} := \mathsf{OT}_3(\mathsf{ot}_2, (\gamma, \mu))$.

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

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

[a] Here, we have to ensure that the party acts $(n-1)$-times as party 1 in Γ, $(n-1)$-times as party 2, and takes the role of one of the remaining parties in the other executions of Γ.

Figure 4.4: Circuit $C^{i,t}$

Input. st

Hard-coded Information. v_i, $\{\mu^{i,t,\alpha,\beta}\}_{\alpha,\beta}$, $\{\mathsf{ot}_2^{t,\alpha,\beta}\}_{\alpha,\beta}$ and $\mathsf{lab} = \{\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}\}_{k \in [\ell]}$.

- Let $\phi_t = (i^*, f, g, h)$.
- If $i = i^*$,then:
  - Compute $\mathsf{st}[h] := \mathsf{NAND}(\mathsf{st}[f] \oplus v_i[f], \mathsf{st}[g] \oplus v_i[g]) \oplus v_i[h]$.
  - Output $((\mathsf{st}[f], \mathsf{st}[g], \mathsf{st}[h]), \mu^{i,t,\mathsf{st}[f],\mathsf{st}[g]}, \{\mathsf{lab}_{k,\mathsf{st}[k]}\}_{k \in [\ell]})$.
- Else:
  - Output $(\mathsf{ot}_2^{i^*,t,\mathsf{st}[f],\mathsf{st}[g]}, \{\mathsf{lab}_{k,\mathsf{st}[k]}\}_{k \in [\ell] \setminus \{h\}})$.

Theorem 4.4. *Let Γ be the protocol for the double selection functionality defined in Section 4.4, OT be the oblivious transfer protocol with equivocal receiver security defined in Section 4.3 and let Φ be a conforming protocol with statistical security in the OT correlation model defined in Section 4.5, then Π satisfies Definition 4.1 and makes black box use of Γ, Φ and OT.*

Proof. To prove the security of the protocol, we define the simulator below. Before describing the simulator, we introduce an additional procedure called $\mathsf{Faithful}$:

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

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

Figure 4.5: Simulator of the Inner-Protocol

Round 1.

$\mathsf{Sim}_\Pi(1^\lambda)$:

1. Initialize a list $\mathsf{aux} = \perp$. This list contains the input and output of every corrupted party given to each invocation of the OT with an honest party in the correlation generation phase.
2. For all $i \in H$, execute the protocol Γ $(n-1)^2\kappa$-times. In the k-th execution (with $k \in [\kappa]$), we denote the message $\gamma_{1,i}^{j,l,k}$ as the message generated in the setting where P_j behaves as party P_1 and P_l behaves as party P_2, sent by

party i. To generate these messages the simulator Sim_Π behaves for all $i \in H$ as follows (namely it uses the honest party procedure of $\mathbf{\Gamma}$):

(a) For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma_{1,i}^{i,j,k} \leftarrow \mathbf{\Gamma}_1(1^\lambda, 1, (\alpha_k^{i,j}, r_k^{i,j}))$, on behalf of P_1, using $\alpha_k^{i,j}$, a uniformly chosen bit, $r_k^{i,j} := v_i[\mathsf{GetIndex}(i,j,k)]$ and $(\mathsf{sk}_{k,0}^{i,i,j}, \mathsf{sk}_{k,1}^{i,i,j})$ are generated using $\mathsf{Gen}(1^\lambda)$.

(b) For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma_{1,i}^{j,i,k} \leftarrow \mathbf{\Gamma}_1(1^\lambda, 2, (y_{k,0}^{j,i}, y_{k,1}^{j,i}))$, on behalf of P_2, using uniformly chosen bits $y_{k,0}^{j,i}, y_{k,1}^{j,i}$ and $(\mathsf{sk}_{k,0}^{i,j,i}, \mathsf{sk}_{k,1}^{i,j,i})$ are generated using $\mathsf{Gen}(1^\lambda)$.

(c) For all $j, l \in [n]$ with $l \neq j$, generate $\gamma_{1,i}^{j,l,k} \leftarrow \mathbf{\Gamma}_1(1^\lambda, i, (\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l}))$ with $\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l}$ being generated using $\mathsf{Gen}(1^\lambda)$.[a]

3. Output $\mathsf{msg}^1_{\Gamma,\mathcal{H}} := \{\{\{\gamma_{1,i}^{i,j,k}, \gamma_{1,i}^{j,i,k}\}_{j\in[n]\setminus\{i\}}, \{\gamma_{1,i}^{j,l,k}\}_{j,l\in[n],l\neq j}\}_{k\in[\kappa]}\}_{i\in H}$.

Receive $\mathsf{msg}^1_{\Gamma,\mathcal{A}} := \{\{\{\gamma_{1,i}^{i,j,k}, \gamma_{1,i}^{j,i,k}\}_{j\in[n]\setminus\{i\}}, \{\gamma_{1,i}^{j,l,k}\}_{j,l\in[n],l\neq j}\}_{k\in[\kappa]}\}_{i\in M}$ from $\mathcal{A}$.

Round 2.

$\mathsf{Sim}_\Pi(1^\lambda)$**:**

1. For each $i \in H$ 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: $(\mathsf{ot}_1^{i,t,\alpha,\beta}, \mu_0^{i,t,\alpha,\beta}, \mu_1^{i,t,\alpha,\beta}) \leftarrow \mathsf{Sim}_{Eq}^{\mathsf{OT}}(1^\lambda)$
2. Continue the execution of the $(n-1)^2\kappa$ instances of the protocol $\mathbf{\Gamma}$. The messages in those executions are generated by the simulator Sim_Π as follows for all $i \in H$:
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma_{2,i}^{i,j,k} \leftarrow \mathbf{\Gamma}_2(1^\lambda, 1, (\alpha_k^{i,j}, r_k^{i,j}))$, on behalf of P_1.
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma_{2,i}^{j,i,k} \leftarrow \mathbf{\Gamma}_2(1^\lambda, 2, (y_{k,0}^{j,i}, y_{k,1}^{j,i}))$, on behalf of P_2.
   - For all $j, l \in [n]$ with $l \neq j$, generate $\gamma_{2,i}^{j,l,k} \leftarrow \mathbf{\Gamma}_2(1^\lambda, i, (\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l}))$.
3. For all $i \in H$, set $z_i := r_i \| 0^{\ell/n-m}$ with r_i being randomly chosen from $\{0,1\}^m$ where m is the total length of the inputs of P_i (the actual input and the OT correlation).
4. For all $i \in H$, set $z_i^{\mathsf{part}} = z_i[(n-1)\kappa + 1, \ell/n]$.
5. Output $\mathsf{msg}^2_{\mathcal{H}} := \{\{\mathsf{ot}_1^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}, z_i^{\mathsf{part}}\}_{i\in H}$ and $\mathsf{msg}^2_{\Gamma,\mathcal{H}} := \{\{\{\gamma_{2,i}^{i,j,k}, \gamma_{2,i}^{j,i,k}\}_{j\in[n]\setminus\{i\}}, \{\gamma_{2,i}^{j,l,k}\}_{j,\ell\in[n],l\neq j}\}_{k\in[\kappa]}\}_{i\in H}$.

Receive $\mathsf{msg}^2_{\mathcal{A}} := \{\{\mathsf{ot}_1^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}\}_{i\in M}$, $\widetilde{\mathsf{msg}}^2_{\mathcal{A}} := \{z_i^{\mathsf{part}}\}_{i\in M}$, $\mathsf{msg}^2_{\Gamma,\mathcal{A}} := \{\{\{\gamma_{2,i}^{i,j,k}, \gamma_{2,i}^{j,i,k}\}_{j\in[n]\setminus\{i\}}, \{\gamma_{2,i}^{j,l,k}\}_{j,l\in[n],l\neq j}\}_{k\in[\kappa]}\}_{i\in M}$, as well as $(r_i)_{i\in M}$ from $\mathcal{A}$.

Round 3. On receiving the random tapes $(R_i)_{i\in M}$ from $\mathcal{A}$ we distinguish between two cases:

(a) At least one of the messages $(\mathsf{msg}^1_{\Gamma,\mathcal{A}}, \mathsf{msg}^2_{\Gamma,\mathcal{A}}, \mathsf{msg}^2_{\mathcal{A}})$ is inconsistent w.r.t. the random tapes $(R_i)_{i\in M}$ provided by $\mathcal{A}$:

1. Sample v_i randomly.
2. Set $v_i[(i-1)\ell/n + (n-1)\kappa + 1, i\ell/n] := z_i^{\mathsf{part}} \oplus x_i^{\mathsf{part}}$.

3. For each $i \in H$ and for each t such that $\phi_t = (i, f, g, h)$ (A_i is the set of such values of t), for each $\alpha, \beta \in \{0,1\}$, $\mu^{i,t,\alpha,\beta} := \mu^{i,t,\alpha,\beta}_{v_i[h] \oplus \mathsf{NAND}(v_i[f] \oplus \alpha, v_i[g] \oplus \beta)}$
4. Set $\mathsf{lab}^{i,T+1} := \{\mathsf{lab}^{i,T+1}_{k,0}, \mathsf{lab}^{i,T+1}_{k,1}\}_{k \in [\ell]}$ where for each $k \in [\ell]$ and $b \in \{0,1\}, \mathsf{lab}^{i,T+1} := \perp$.
5. For each t from T down to 1, do the following:
   (a) Let ϕ_t as (i^*, f, g, h).
   (b) If $i = i^*$, compute $(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \{\mu^{i,t,\alpha,\beta}\}_{\alpha,\beta}, \perp, \mathsf{lab}^{i,t+1}])$.
   (c) If $i \neq i^*$, then for every $\alpha, \beta \in \{0,1\}$, set $\mathsf{ot}_2^{i^*,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{ot}_1^{i^*,t,\alpha,\beta}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$ and compute $(\widetilde{C}^{i,t}, \mathsf{lab}^{i,t}) \leftarrow \mathsf{Garble}(1^\lambda, C^{i,t}[v_i, \perp, \{\mathsf{ot}_2^{i,t,\alpha,\beta}\}_{\alpha,\beta}, \mathsf{lab}^{i,t+1}])$.
6. For each $j, j' \in [n]$ with $j \neq j'$ and $k \in [\kappa]$, compute $\mathsf{ct}^{i,j,j'}_{k,0} = \mathsf{Enc}(\mathsf{sk}^{i,j,j'}_{k,0}, \mathsf{lab}^{i,1}_{\mathsf{GetIndex}(j,j',k),0})$ and $\mathsf{ct}^{i,j,j'}_{k,1} = \mathsf{Enc}(\mathsf{sk}^{i,j,j'}_{k,1}, \mathsf{lab}^{i,1}_{\mathsf{GetIndex}(j,j',k),1})$.
7. Continue the execution of the $(n-1)^2\kappa$ instances of the protocol $\mathbf{\Gamma}$. The messages in those executions are generated as follows:
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma^{i,j,k}_{3,i} \leftarrow \mathbf{\Gamma}_3(1^\lambda, 1, (\alpha^{i,j}_k, r^{i,j}_k))$, on behalf of P_1.
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma^{j,i,k}_{3,i} \leftarrow \mathbf{\Gamma}_3(1^\lambda, 2, (y^{j,i}_{k,0}, y^{j,i}_{k,1}))$, on behalf of P_2.
   - For all $j, l \in [n]$ with $l \neq j$, generate $\gamma^{j,l,k}_{3,i} \leftarrow \mathbf{\Gamma}_3(1^\lambda, i, (\mathsf{sk}^{i,j,l}_{k,0}, \mathsf{sk}^{i,j,l}_{k,1}))$.
8. Set $\mathsf{st} := \Big((0^{(n-1)\kappa} \| z_1^{\mathsf{part}}) \| \ldots \| (0^{(n-1)\kappa} \| z_n^{\mathsf{part}})\Big)$.
9. Output $\mathsf{msg}^3_{\mathcal{H}} := (\{\widetilde{C}^{i,t}\}_{t \in [T]}, (\mathsf{ct}^{i,j,j'}_{k,0}, \mathsf{ct}^{i,j,j'}_{k,1})_{j,j' \in [n], j \neq j', k \in [\kappa]}, \{\mathsf{lab}^{i,1}_{k,\mathsf{st}[k]}\}_{k \notin [(j-1)\ell/n+1, (j-1)\ell/n+(n-1)\kappa]})_{i \in \mathcal{H}}$ and $\mathsf{msg}^3_{\Gamma,\mathcal{H}} := \{\{\{\gamma^{i,j,k}_{3,i}, \gamma^{j,i,k}_{3,i}\}_{j \in [n] \setminus \{i\}}, \{\gamma^{j,l,k}_{3,i}\}_{j,l \in [n], l \neq j}\}_{k \in [\kappa]}\}_{i \in \mathcal{H}}$.

(b) All the messages $(\mathsf{msg}^1_{\mathit{\Gamma},\mathcal{A}}, \mathsf{msg}^2_{\mathit{\Gamma},\mathcal{A}}, \mathsf{msg}^2_{\mathcal{A}})$ are consistent w.r.t. random tapes $\{R_i\}_{i \in M}$ provided by $\mathcal{A}$:

$\mathsf{Sim}_\Pi(1^\lambda, \mathsf{aux})$**:** 1. Set $\mathsf{st}^* := \Big((0^{(n-1)\kappa} \| z_1^{\mathsf{part}}) \| \ldots \| (0^{(n-1)\kappa} \| z_n^{\mathsf{part}})\Big)$.

2. For each $j \in H$ and for each $k \in [(j-1)\ell/n + 1, (j-1)\ell/n + (n-1)\kappa]$, set $\mathsf{st}^*[k] = z_j[k - (j-1)\ell/n]$.
3. For every $i \in M$:
   (a) Using messages $\mathsf{msg}^1_{\mathit{\Gamma},\mathcal{A}}, \mathsf{msg}^2_{\mathit{\Gamma},\mathcal{A}}$ and randomness $(R_i)_{i \in M}$ compute $(\alpha^{i,j}_k, r^{i,j}_k)$ for all $i \in M, j \in H, k \in [\kappa]$. Afterwards, add $((i,j,k), (\alpha^{i,j}_k, y^{i,j}_{k,\alpha^{i,j}_k}))$ to aux for all $i \in M, j \in H, k \in [\kappa]$, where $y^{i,j}_{k,\alpha^{i,j}_k}$ is chosen uniformly at random in Round 1.
   (b) Using messages $\mathsf{msg}^1_{\mathit{\Gamma},\mathcal{A}}, \mathsf{msg}^2_{\mathit{\Gamma},\mathcal{A}}$ and randomness $(R_i)_{i \in M}$ compute $((i,j,k), (y^{j,i}_{k,0}, y^{j,i}_{k,1}))$ for all $i \in H, j \in M, k \in [\kappa]$. Afterwards, add $((i,j,k), (y^{j,i}_{k,0}, y^{j,i}_{k,1}))$ to aux for all $i \in H, j \in M, k \in [\kappa]$.
   (c) For every $j \in H$, $j \neq i$ and for each $k \in [\kappa]$ set $\mathsf{st}^*[\mathsf{GetIndex}(i,j,k)] := y^{i,j}_{k,\alpha^{i,j}_k} \oplus r^{i,j}_k$.

   (d) For every $j \in M$, $j \neq i$ and for each $k \in [\kappa]$, using messages $\mathsf{msg}^1_{\Gamma,\mathcal{A}}, \mathsf{msg}^2_{\Gamma,\mathcal{A}}$ and randomness $(R_i)_{i \in M}$ to compute $\beta^{i,j}_k$ and set $\mathsf{st}^*[\mathsf{GetIndex}(i,j,k)] := \beta^{i,j}_k$.

4. Use the randomness $(R_i)_{i \in M}$ to compute $b^{i,t,\alpha,\beta}$ as the input used for $\mathsf{ot}_1^{i,t,\alpha,\beta}$ for all $t \in A_i, \alpha, \beta \in \{0,1\}$.
5. Initialize the simulator Sim_Φ of the conforming protocol Φ with the values $(H, \mathsf{st}^*, \mathsf{aux})$ to start the computation phase of Φ with the simulator Sim_Φ.
6. The messages for Sim_Φ for the corrupted parties $i \in M$ are simulated using the $\mathsf{Faithful}(i, \mathsf{st}^*, \{b^{i,t,\alpha,\beta}\}_{t \in A_i, \alpha, \beta})$ execution, i.e., for all $i \in M$, the procedure $\mathsf{Faithful}$ is executed and the resulting messages are used as an input to Sim_Φ.
7. Once the simulator Sim_Φ submits the ideal functionality query $\{x_i\}_{i \in M}$, this query is forwarded to the ideal functionality, to obtain as an output the function evaluation out. This, in turn, is forwarded as a reply to the simulator Sim_Φ. Sim_Φ then continues the simulation of the conforming protocol.
8. Let $\mathsf{Z} \in \{0,1\}^t$ where Z_t is the bit sent in the t'th round of the computation phase of Φ, and let st^*_T be the state value at the end of the faithful execution of one of the corrupted parties (this value is the same for all the parties). For each $t \in \cup_{i \in H} A_i$ and $\alpha, \beta \in \{0,1\}$, set $\mu^{i,t,\alpha,\beta} := \mu^{i,t,\alpha,\beta}_{\mathsf{Z}_t}$.
9. Set $\mathsf{lab}^{i,T+1} := \{\mathsf{lab}^{i,T+1}_{k,0}, \mathsf{lab}^{i,T+1}_{k,1}\}_{k \in [\ell]}$ where for each $k \in [\ell]$ and $b \in \{0,1\}$, $\mathsf{lab}^{i,T+1} := \bot$.
10. For each t from T down to 1, do the following:
   (a) Parse ϕ_t as (i^*, f, g, h).
   (b) Set $\alpha^* := \mathsf{st}^*_T[f], \beta^* := \mathsf{st}^*_T[g]$, and $\gamma^* := \mathsf{st}^*_T[h]$.
   (c) If $i = i^*$, compute $(\widetilde{C}^{i,t}, \{\mathsf{lab}^{i,t}_{k,\mathsf{st}^*_T[k]}\}_{k \in [\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \big((\alpha^*, \beta^*, \gamma^*), \mu^{i^*,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_k\}_{k \in [\ell]}\big))$.
   (d) If $i \neq i^*$, then set $\mathsf{ot}_2^{i,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2(\mathsf{ot}_1^{i^*,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_h, \mathsf{lab}^{i,t+1}_h)$ and compute $\mathsf{ot}_2^{i^*,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{ot}_1^{i^*,t,\alpha,\beta}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$ and compute $(\widetilde{C}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k \in [\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \big(\mathsf{ot}_2^{i,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_k\}_{k \in [\ell] \setminus \{h\}}\big))$.
11. For each $j, j' \in [n]$ with $j \neq j'$ and $k \in [\kappa]$, compute $\mathsf{ct}^{i,j,j'}_{k,0} = \mathsf{Enc}(\mathsf{sk}^{i,j,j'}_{k,0}, \mathsf{lab}^{i,1}_{\mathsf{GetIndex}(j,j',k)})$ and $\mathsf{ct}^{i,j,j'}_{k,1} = \mathsf{Enc}(\mathsf{sk}^{i,j,j'}_{k,1}, \mathsf{lab}^{i,1}_{\mathsf{GetIndex}(j,j',k)})$.
12. Continue the execution of the $(n-1)^2\kappa$ instances of the protocol $\mathbf{\Gamma}$. The messages in those executions are generated as follows:
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma^{i,j,k}_{3,i} \leftarrow \mathbf{\Gamma}_3(1^\lambda, 1, (\alpha^{i,j}_k, r^{i,j}_k))$, on behalf of P_1.
   - For all $j \in [n] \setminus \{i\}, k \in [\kappa]$, generate $\gamma^{j,i,k}_{3,i} \leftarrow \mathbf{\Gamma}_3(1^\lambda, 2, (y^{j,i}_{k,0}, y^{j,i}_{k,1}))$, on behalf of P_2.
   - For all $j, l \in [n]$ with $l \neq j$, generate $\gamma^{j,l,k}_{3,i} \leftarrow \mathbf{\Gamma}_3(1^\lambda, i, (\mathsf{sk}^{i,j,l}_{k,0}, \mathsf{sk}^{i,j,l}_{k,1}))$.

13. Output $\mathsf{msg}^3_{\mathcal{H}} := (\{\widetilde{C}^{i,t}\}_{t\in[T]}, (\mathsf{ct}^{i,j,j'}_{k,0}, \mathsf{ct}^{i,j,j'}_{k,1})_{j,j'\in[n],j\neq j',k\in[\kappa]}, \{\mathsf{lab}^{i,1}_{k,\mathsf{st}[k]}\}_{k\notin[(j-1)\ell/n+1,(j-1)\ell/n+(n-1)\kappa]})_{i\in\mathcal{H}}$ and $\mathsf{msg}^3_{\Gamma,\mathcal{H}} := \{\{\{\gamma^{i,j,k}_{3,i}, \gamma^{j,i,k}_{3,i}\}_{j\in[n]\setminus\{i\}}, \{\gamma^{j,l,k}_{3,i}\}_{j,l\in[n],l\neq j}\}_{k\in[\kappa]}\}_{i\in\mathcal{H}}$.

Output Computation. Obtain the third round messages for all $i \in M$ and check if the execution of the garbled succeeds. If the computation succeeds, then Sim computes the output of the honest parties otherwise Sim sets the output of the honest parties to $\perp$. Sim outputs the view of $\mathcal{A}$ in the above execution and the outputs of the honest parties.

[a] Here, we have to ensure that the party acts $(n-1)$-times as party 1 in Γ, $(n-1)$-times as party 2, and takes the role of one of the remaining parties in the other executions of Γ.

Before beginning to prove the semi-malicious security of the protocol, we observe that the equivocal security of the protocol directly follows from the analysis presented in [IKSS23, Section 8.4] since their analysis for the case of equivocal security corresponds to the same simulator as the one described above.

Furthermore, it follows by inspection that the operations computed by the algorithm Π_2^{inp} do not involve any cryptographic computation, which covers the second property of the security definition.

Now, we proceed with the hybrids used to prove the semi-malicious security (the proof proceeds similarly to [IKSS23] and [PS21]):

Hybrid H_0: This hybrid corresponds to the real-world experiment. The output of the hybrid is the view of $\mathcal{A}$ in the this execution and the outputs of the honest parties.

Hybrid H_1: This hybrid is defined as the previous one but instead of executing the protocol Γ honestly, we run it in equivocation mode. In more detail, the first round messages $\mathsf{msg}^1_{\Gamma,\mathcal{H}}$ are generated using $\mathsf{Sim}_\Gamma(1^\lambda, i)$ for all $i \in H$ and the remaining messages are generated using Equiv instead of $\Gamma_j(1^\lambda, i, \cdot)$ for all $i \in H, j \in \{2,3\}$. The indistinguishability between this and the previous hybrid follows from the security of Γ, which we argue formally in Lemma 4.5.

Hybrid H_2: This hybrid is defined as the previous one but runs in exponential time to extract the inputs $\{x_i\}_{i\in H}$ of the malicious parties in the first round $\mathsf{msg}^1_{\Gamma,\mathcal{A}}$ of Γ. Then, the hybrid runs $\mathsf{Sim}_{\Gamma_j}(1^\lambda, i, \{x_i\}_{i\in H})$, for all $i \in H, j \in \{2,3\}$, to simulate the second $\mathsf{msg}^2_{\Gamma,\mathcal{H}}$ and third round $\mathsf{msg}^3_{\Gamma,\mathcal{H}}$ of Γ on behalf of the honest parties. The indistinguishability between this and the previous hybrid follows from the security of Γ, which we argue more formally in Lemma 4.6.

Hybrid H_3: This hybrid is defined as the previous one but we change the way in which the messages $\mathsf{ot}_2^{i^*,t,\alpha,\beta}$ inside the garbled circuits are generated. In more detail, for all $i \in H$, for each $t \in [T]$, and $\alpha, \beta \in \{0,1\}$ let $\phi_t = (i^*, f, g, h)$, then for all $i \neq i^*$, $\mathsf{ot}_2^{i,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2(\mathsf{ot}_1^{i^*,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{b^{i,t,\alpha,\beta}}, \mathsf{lab}^{i,t+1}_{b^{i,t,\alpha,\beta}})$ where $b^{i,t,\alpha,\beta}$ is the input obtained from the messages of the adversary as described in the simulation strategy, if $i^* \in M$ and $b^{i,t,\alpha,\beta} := v_{i^*}[h] \oplus \mathsf{NAND}(v_{i^*}[f] \oplus \alpha, v_{i^*}[g] \oplus \beta)$, if $i^* \in H$. The indistinguishability between this and the previous hybrid follows from the sender privacy of the OT, which we argue more formally in Lemma 4.7.

Hybrid H_4: This hybrid is defined as the previous one but we generate the OT messages $\mathsf{ot}_1^{i,t,\alpha,\beta}$ using the equivocator. In more detail, for all $i \in H$ and for each $t \in A_i$ and $\alpha, \beta \in \{0,1\}$, the messages $\mathsf{ot}_1^{i,t,\alpha,\beta}$ are generated using $(\mathsf{ot}_1^{i,t,\alpha,\beta}, \mu_0^{i,t,\alpha,\beta}, \mu_1^{i,t,\alpha,\beta}) \leftarrow \mathsf{Sim}_{Eq}^{\mathsf{OT}}(1^\lambda)$ instead of $(\mathsf{ot}_1^{i,t,\alpha,\beta}, \mu^{i,t,\alpha,\beta}) \leftarrow \mathsf{OT}_1(v_i[h] \oplus \mathsf{NAND}(v_i[f] \oplus \alpha, v_i[g] \oplus \beta))$. The indistinguishability between this and the previous hybrid follows from the equivocal receiver security of the OT, which we argue more formally in Lemma 4.8.

Hybrid H_5: This hybrid is defined as the previous one but we generate the state st^* and initialize the list aux with respect to the values used in the protocol $\mathbf{\Gamma}$. This happens almost as described in the simulator above (Step 2(b) of round 1 and step 1-3 of round 3). In more detail:

1. Initialize $\mathsf{aux} = \perp$.
2. Set $\mathsf{st}^* = \Big((0^{(n-1)\kappa} \| z_1^{\mathsf{part}}) \| \ldots \| (0^{(n-1)\kappa} \| z_n^{\mathsf{part}})\Big)$.
3. For every $i \in M$:
   - For every $j \in H$ with $j \neq i$ and for each $k \in [\kappa]$:
     (a) Compute $\alpha_k^{i,j}, r_k^{i,j}$ using the messages and the randomness of the adversary as described in the simulation strategy.
     (b) Choose a random bit $y_{k,\alpha_k^{i,j}}^{i,j}$.
     (c) Set $\mathsf{st}^*[\mathsf{GetIndex}(i,j,k)] \coloneqq y_{k,\alpha_k^{i,j}}^{i,j} \oplus r_k^{i,j}$.
     (d) Add $((i,j,k), (\alpha_k^{i,j}, y_{k,\alpha_k^{i,j}}^{i,j}))$ to aux.
   - For every $j \in M$ with $j \neq i$ and for each $k \in [\kappa]$:
     (a) Set $\mathsf{st}^*[\mathsf{GetIndex}(i,j,k)] \coloneqq \beta_k^{i,j}$.
4. For each $i \in H$:
   (a) Compute $(y_{k,0}^{i,j}, y_{k,1}^{i,j})$ using the messages and the randomness of the adversary as described in the simulation strategy.
   (b) Set $\mathsf{st}^*[\mathsf{GetIndex}(i,j,k)] \coloneqq y_{k,\alpha_k^{i,j}}^{i,j} \oplus r_k^{i,j}$ where $\alpha_k^{i,j}, r_k^{i,j}$ and $(y_{k,0}^{i,j}, y_{k,1}^{i,j})$ (if $j \in H$) are computed using the honest parties randomness.
   (c) Add $((i,j,k), (y_{k,0}^{i,j}, y_{k,1}^{i,j}))$ to aux if $j \in M$.

From the construction of the state st^*, it follows that it is consistent with the adversarial and the honest parties input/randomness in the correlations phase. The same holds for the list aux which contains the inputs and outputs of the adversarial parties during OT invocations with an honest party in the correlations phase. Since this hybrid does not change the distributions of the messages with respect to the previous hybrid, those hybrids are identical.

Hybrid H_6: This hybrid is defined as the previous one but we change the encryptions of the labels for the garbled circuits that are output in the third round of the protocol. In more detail, the hybrid proceeds as follow: For all $j, j' \in [n]$ with $j \neq j'$ and $k \in [\kappa]$ on behalf of all the honest parties $i \in H$

1. Set $\eta = \mathsf{GetIndex}(j, j', k)$
2. Compute and ouput $\mathsf{ct}_{k,0}^{i,j,j'} = \mathsf{Enc}(\mathsf{sk}_{k,0}^{i,j,j'}, \mathsf{lab}_{\eta,\mathsf{st}^*[\eta]}^{i,1})$ and $\mathsf{ct}_{k,1}^{i,j,j'} = \mathsf{Enc}(\mathsf{sk}_{k,1}^{i,j,j'}, \mathsf{lab}_{\eta,\mathsf{st}^*[\eta]}^{i,1})$

The indistinguishability between this and the previous hybrid follows from the semantic security of the symmetric encryption scheme, which we argue more formally in Lemma 4.9.

Hybrid H_7: This hybrid is defined as the previous one but the garbled circuits for all the T different steps of the conforming protocol Φ are simulated instead of honestly generated. In more detail, we introduce the intermediate hybrids $\mathsf{H}_{7,t}$ for $t \in \{0, \ldots, T\}$ where the distribution of $\mathsf{H}_{7,t}$ is the same as the one of $\mathsf{H}_{7,t-1}$ except for the garbled circuits (in the third round) that play a role in the execution of the t'th round of the protocol Φ; namely, the action $\phi_t = (i^*, f, g, h)$. Here, the execution of Φ is completed using the honest parties inputs and randomness. The messages on behalf of the the corrupted parties are generated via faithful execution, i.e., by executing $\mathsf{Faithful}(i, \mathsf{st}^*, \{b^{i,t,\alpha,\beta}\}_{t \in A_i, \alpha, \beta})$ for the malicious parties. Let $\mathsf{Z} \in \{0,1\}^T$ be the transcript obtained using the above step. Let st^*_T be the public state of one of the corrupted parties a the end of faithful execution and let st^*_t be the public state of the parties at the end of the t'th round of the computation phase. Finally, let $\alpha^* := \mathsf{st}^*_T[f], \beta^* := \mathsf{st}^*_T[g]$ and $\gamma^* := \mathsf{st}^*_T[h]$. In $\mathsf{H}_{7,t}$ the following changes are made with respect to hybrid $\mathsf{H}_{7,t-1}$:

1. We make the following two changes in how we generate messages for other honest parties $i \in H \setminus \{i^*\}$. We do not generate four $\mathsf{ot}_2^{i^*,t,\alpha,\beta}$ values but just one of them; namely, we generate $\mathsf{ot}_2^{i^*,t,\alpha^*,\beta^*}$ as $\mathsf{OT}_2(\mathsf{ot}_1^{i^*,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t})$. Second, we generate the garbled circuit
$$(\widetilde{C}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k \in [\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left(\mathsf{ot}_2^{i^*,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k \in [\ell] \setminus \{h\}}\right)\right).$$
2. If $i^* \in M$ then skip these changes. We make two changes in how we generate messages on behalf of i^*. First, for all $\alpha, \beta \in \{0,1\}$, we set $\mu^{i^*,t,\alpha^*,\beta^*}$ as $\mu^{i^*,t,\alpha^*,\beta^*}_{\mathsf{Z}_t}$ rather than $\mu^{i^*,t,\alpha^*,\beta^*}_{v_{i^*}[h] \oplus \mathsf{NAND}(v_{i^*}[f] \oplus \alpha^*, v_{i^*}[g] \oplus \beta^*)}$ (note that these two values are the same when using the honest party's input and randomness). Second, it generates the garbled circuit
$$(\widetilde{C}^{i^*,t}, \{\mathsf{lab}^{i^*,t}_k\}_{k \in [\ell]}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}\left(1^\lambda, 1^{|C^{i,t}|}, 1^\ell, \left((\alpha^*, \beta^*, \gamma^*), \mu^{i^*,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i^*,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k \in [\ell]}\right)\right).$$

The labels $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k \in [\ell]} / \{\mathsf{lab}^{i^*,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k \in [\ell]}$ here are the honestly generated input labels for the garbled circuit $\widetilde{C}^{i,t+1} / \widetilde{C}^{i^*,t+1}$ (for any $t \leq T-1$) and for $t = T, \{\mathsf{lab}^{i,T+1}_{k,\mathsf{st}^*_T[k]} := \perp\}_{k \in [\ell]} / \{\mathsf{lab}^{i^*,T+1}_{k,\mathsf{st}^*_T[k]} := \perp\}_{k \in [\ell]}$.
The indistinguishability between $\mathsf{H}_{7,t-1}$ and $\mathsf{H}_{7,t}$ for every $t \in [T]$ follows from the security of the garbled circuit which we prove more formally in Lemma 4.10. This, in turn, then implies the indistinguishability between H_6 and H_7.

Hybrid H_8: This hybrid is defined as the previous one but the output phase of the computation is modified to execute the garbled circuits provided by $\mathcal{A}$ on behalf of the corrupted parties and check if their execution proceeds consistently with the transcript Z. If the computation succeeds then for each $i \in H$, the parties are instructed to output the result of the output computation, else, they are instructed to output $\perp$. The indistinguishability between this and the previous hybrid follows from the authenticity of the input labels property of the garbled circuit.

Hybrid H_9: This hybrid is defined as the previous one but we change the way in which the transcripts $\mathsf{Z}, \{z_i\}_{i \in H}$ and the values st_T^* are generated are changed. Instead of generating these values using the input of the honest party in the faithful execution of Φ, they are generated via the simulator Sim_Φ using the list aux as an additional input. More specifically, z_i is generated as $(r_i \| 0^{\ell/n-m})$ where r_i is a uniformly chosen random string of length m for each $i \in H$. Here, st_T^* is generated as described in the simulator. To generate the transcript of Φ, the simulator Sim_Φ is executed using the input $(H, \mathsf{st}^*, \mathsf{aux})$ where messages on behalf of each corrupted party P_i are generated using $\mathsf{Faithful}(i, \mathsf{st}^*, \{b^{i,t\alpha,\beta}\}_{t \in A_i, \alpha, \beta})$. From the statistical security of Φ it now follows that this hybrid is identically distributed to the previous hybrid, which we argue more formally in Lemma 4.11.

Hybrid H_{10}: This hybrid is defined as the previous one but we invert the change from hybrid H_2. In more detail, instead of simulating the second and third rounds of the protocol Γ, those messages are generated honestly again using a random input. This hybrid is also not executed in exponential time anymore. The indistinguishability of this and the previous hybrid follows analogously to the indistinguishability of H_1 and H_2. We refer to the proof of Lemma 4.6 for further details.

Hybrid H_{11}: This hybrid corresponds to the ideal world. In this hybrid, we also execute the first message of the protocol Γ honestly and not using Equiv. This hybrid inverts the change of H_1 and its indistinguishability to the previous hybrid follows analogously to the indistinguishability of hybrid H_0 and H_1. We refer to the proof of Lemma 4.5 for further details.

□

Lemma 4.5 (Transition from Hybrid H_0 to H_1). *Let Γ be secure as defined in Definition 4.3, then the hybrids H_0 and H_1 are indistinguishable.*

Proof. We prove this lemma by contradiction, we assume that there exists an adversary $\mathcal{A}$ that distinguishes between the hybrids H_0 and H_1 with non-negligible probability. We use this adversary to construct an adversary $\mathcal{B}$ that breaks the security of the protocol Γ with non-negligible probability.

Now, we describe the behavior of the adversary $\mathcal{B}$, which behaves on behalf of the honest parties $i \in H$, interacting with the challenger of the security of the protocol Γ and the adversary $\mathcal{A}$. In the first step, the adversary $\mathcal{B}$ generates $\{(\alpha_k^{i,j}, r_k^{i,j})_{j \in [n] \setminus \{i\}, k \in [\kappa]}, (y_{k,0}^{j,i}, y_{k,1}^{j,i})_{j \in [n] \setminus \{i\}, k \in [\kappa]}, (\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l})_{j,l \in [n], j \neq l, k \in [\kappa]}\}_{i \in H}$ as described in H_0 and send them to the challenger obtaining $\mathsf{msg}^1_{\Gamma,\mathcal{H}}$.

$\mathcal{B}$ start the execution with $\mathcal{A}$ behaving as in the protocol, except for the generation of the messages $\mathsf{msg}^1_{\Gamma,\mathcal{H}}$, which $\mathcal{B}$ receives from the underlying challenger of Γ. Afterwards, the adversary $\mathcal{B}$ receives as a reply the messages $\mathsf{msg}^1_{\Gamma,\mathcal{A}}$ from $\mathcal{A}$. For the second round, the adversary $\mathcal{B}$ behaves again as in the previous hybrid to generate $\mathsf{msg}^2_{\mathcal{H}}$. To obtain the messages $\mathsf{msg}^2_{\Gamma,\mathcal{A}}$, the adversary $\mathcal{B}$ sends $\mathsf{msg}^1_{\Gamma,\mathcal{A}}$ to the underlying challenger which will reply with $\mathsf{msg}^2_{\Gamma,\mathcal{H}}$. The messages $\mathsf{msg}^2_{\mathcal{H}}$ and $\mathsf{msg}^2_{\Gamma,\mathcal{H}}$ are then forwarded to $\mathcal{A}$ which replies with $\mathsf{msg}^2_{\mathcal{A}}, \mathsf{msg}^2_{\Gamma,\mathcal{A}}$. The messages $\mathsf{msg}^2_{\Gamma,\mathcal{A}}$ are used to receive the messages $\mathsf{msg}^3_{\Gamma,\mathcal{H}}$ from the underlying challenger. The messages $\mathsf{msg}^3_{\mathcal{H}}$ are then generated as in the previous hybrid and sent together with $\mathsf{msg}^3_{\mathcal{H}}$ to $\mathcal{A}$. Finally, the adversary $\mathcal{A}$ outputs the messages $\mathsf{msg}^3_{\mathcal{A}}$ and $\mathsf{msg}^3_{\Gamma,\mathcal{A}}$.

We observe that if the underlying challenger generates the messages $\mathsf{msg}^1_{\Gamma,\mathcal{H}}, \mathsf{msg}^2_{\Gamma,\mathcal{H}}$ and $\mathsf{msg}^3_{\Gamma,\mathcal{H}}$ honestly, then the adversary $\mathcal{B}$ simulates the hybrid H_0 towards $\mathcal{A}$ and if the underlying challenger generates these messages using Equv then the adversary $\mathcal{B}$ simulates the hybrid H_1 towards $\mathcal{A}$. This directly results in the fact that if $\mathcal{A}$ can distinguish the hybrids H_0 and H_1 with non-negligible probability then $\mathcal{B}$ can also break the security of the protocol Γ with non-negligible probability. This concludes the proof of the lemma. □

Lemma 4.6 (Transition from Hybrid H_1 to H_2). *Let Γ be secure, then the hybrids H_1 and H_2 are indistinguishable.*

Proof. We prove this lemma by contradiction, we assume that there exists an adversary $\mathcal{A}$ that distinguishes between the hybrids H_1 and H_2 with non-negligible probability. We use this adversary to construct an adversary $\mathcal{B}$ that breaks the security of the protocol Γ with non-negligible probability.

Since in both the hybrids the first round of the protocol is identically distributed then we can non-uniformly fix it, and therefore we can also non-uniformly fix the input and randomness used by the adversary in the first round (a similar argument was used in [IKSS23]). Since these values are fixed we can set them as an auxiliary input of the reduction, namely the auxiliary input of $\mathcal{B}$ is $(\alpha^{i,j}_k, r^{i,j}_k)_{j\in[n]\setminus\{i\},k\in[\kappa]}, (y^{j,i}_{k,0}, y^{j,i}_{k,1})_{j\in[n]\setminus\{i\},k\in[\kappa]}$ and $(\mathsf{sk}^{i,j,l}_{k,0}, \mathsf{sk}^{i,j,l}_{k,1})_{j,l\in[n],j\neq l,k\in[\kappa]}$ for all $i \in M$ which are the values used for the generation of $\mathsf{msg}^1_{\Gamma,\mathcal{A}}$.

Now, we describe the behavior of the adversary $\mathcal{B}$, which behaves on behalf of the honest parties $i \in H$, interacting with the challenger of the security of the protocol Γ and the adversary $\mathcal{A}$. In the first step, the adversary $\mathcal{B}$ behaves as in the protocol, starting from the second round. For the second round, the adversary $\mathcal{B}$ behaves as in the previous hybrid to generate $\mathsf{msg}^2_{\mathcal{H}}$. To obtain the messages $\mathsf{msg}^2_{\Gamma,\mathcal{A}}$, the adversary $\mathcal{B}$ sends $(\{(\alpha^{i,j}_k, r^{i,j}_k)_{j\in[n]\setminus\{i\},k\in[\kappa]}, (y^{j,i}_{k,0}, y^{j,i}_{k,1})_{j\in[n]\setminus\{i\},k\in[\kappa]}, (\mathsf{sk}^{i,j,l}_{k,0}, \mathsf{sk}^{i,j,l}_{k,1})_{j,l\in[n],j\neq l,k\in[\kappa]}\}_{i\in H},$ $\{\{(z_{\mathsf{GetIndex}(i,j,k)}, \{\mathsf{sk}^{s,j,j'}\}_{s\in[n]})\}_{j,j'\in[n],j\neq j',k\in[\kappa]}\}_{i\in[n]})$, computed using the auxiliary input, as well as $\mathsf{msg}^1_{\Gamma,\mathcal{A}}$ to the underlying challenger which will reply with $\mathsf{msg}^2_{\Gamma,\mathcal{H}}$. The messages $\mathsf{msg}^2_{\mathcal{H}}$ and $\mathsf{msg}^2_{\Gamma,\mathcal{H}}$ are then forwarded to $\mathcal{A}$ which replies with $\mathsf{msg}^2_{\mathcal{A}}, \mathsf{msg}^2_{\Gamma,\mathcal{A}}$. The messages $\mathsf{msg}^2_{\Gamma,\mathcal{A}}$ are used to receive the messages $\mathsf{msg}^3_{\Gamma,\mathcal{H}}$ from the underlying challenger. The messages $\mathsf{msg}^3_{\mathcal{H}}$ are then generated as in the previous hybrid and sent together with $\mathsf{msg}^3_{\mathcal{H}}$ to $\mathcal{A}$. Finally, the adversary $\mathcal{A}$ outputs the messages $\mathsf{msg}^3_{\mathcal{A}}$ and $\mathsf{msg}^3_{\Gamma,\mathcal{A}}$.

We observe that if the underlying challenger generates the messages $\mathsf{msg}^2_{\Gamma,\mathcal{H}}$ and $\mathsf{msg}^3_{\Gamma,\mathcal{H}}$ using Equiv, then the adversary $\mathcal{B}$ simulates the hybrid H_1 towards $\mathcal{A}$ and if the underlying challenger generates these messages using the simulator then the adversary $\mathcal{B}$ simulates the hybrid H_2 towards $\mathcal{A}$. This directly results in the fact that if $\mathcal{A}$ can distinguish the hybrids H_1 and H_2 with non-negligible probability then $\mathcal{B}$ can also break the security of the protocol Γ with non-negligible probability. This concludes the proof of the lemma. □

Lemma 4.7 (Transition from Hybrid H_2 to H_3). *Let OT be sender private, then the hybrids H_2 and H_3 are indistinguishable.*

Proof. We prove this lemma by contradiction, we assume that there exists an adversary $\mathcal{A}$ that distinguishes between the hybrids H_3 and H_4 with non-negligible probability. We use this adversary to construct an adversary $\mathcal{B}$ that breaks the sender privacy of the OT protocol OT with non-negligible probability.

Using the same argument as in the transition from hybrid H_1 to H_2 (Lemma 4.6), also here $\mathcal{B}$ receives as an auxiliary input the values $(\alpha_k^{i,j}, r_k^{i,j})_{j\in[n]\setminus\{i\},k\in[\kappa]}$, $(y_{k,0}^{j,i}, y_{k,1}^{j,i})_{j\in[n]\setminus\{i\},k\in[\kappa]}$ and $(\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l})_{j,l\in[n],j\neq l,k\in[\kappa]}$ for all $i \in M$ which are the values used for the generation of $\mathsf{msg}^1_{\Gamma,\mathcal{A}}$.

Now, we describe the behavior of the adversary $\mathcal{B}$, which behaves on behalf of the honest parties $i \in H$, interacting with the challenger of the adaptive semi-honest sender security of the OT protocol OT, for all the honest parties $i \in H$, and the adversary $\mathcal{A}$. The adversary behaves exactly as in the previous hybrid to generate the messages $\mathsf{msg}^2_{\Gamma,\mathcal{H}}, \mathsf{msg}^2_{\mathcal{H}}, \mathsf{msg}^3_{\Gamma,\mathcal{H}}$. To generate the messages $\mathsf{ot}_2^{i^*,t,\alpha,\beta}$ for the garbled circuits and labels that are part of the final message $\mathsf{msg}^3_{\mathcal{H}}$, the adversary $\mathcal{B}$ interacts with the underlying challenger. In more detail, for all $i \in H$, for each $t \in [T]$, let $\phi_t = (i^*, f, g, h)$, then for all $i \neq i^*$, $\mathcal{B}$ computes $b^{i,t,\alpha,\beta}$ from the randomness (R_i) of the adversary (as specified in the simulation strategy), if $i^* \in M$ and $b^{i,t,\alpha,\beta} := v_{i^*}[h] \oplus \mathsf{NAND}(v_{i^*}[f] \oplus \alpha, v_{i^*}[g] \oplus \beta)$, if $i^* \in H$. The adversary $\mathcal{B}$ then submits $(b^{i,t,\alpha,\beta}, R_i, \mathsf{ot}_1^{i,t,\alpha,\beta}, \mathsf{lab}_0^{i,t+1}, \mathsf{lab}_1^{i,t+1})$ to the underlying challenger, where $\mathsf{ot}_1^{i,t,\alpha,\beta}$ is part of $\mathsf{msg}^2_{\mathcal{A}}$ received from $\mathcal{A}$. The underlying challenger will reply with $\mathsf{ot}_2^{i,t,\alpha^*,\beta^*}$ which is then used as in the previous hybrid to generate the final message $\mathsf{msg}^3_{\mathcal{H}}$. We observe that if the underlying challenger generates the second round of the OT using the inputs $(\mathsf{lab}_0^{i,t+1}, \mathsf{lab}_1^{i,t+1})$ on behalf of the honest parties then the hybrid H_2 is simulated towards $\mathcal{A}$ and if the underlying challenger generates the OTs using the inputs $(\mathsf{lab}^{i,t+1}_{b^{i,t,\alpha,\beta}}, \mathsf{lab}^{i,t+1}_{b^{i,t,\alpha,\beta}})$ then the hybrid H_3 is simulated towards $\mathcal{A}$. This directly results in the fact that if $\mathcal{A}$ can distinguish the hybrids H_2 and H_3 with non-negligible probability then $\mathcal{B}$ can also break the sender privacy with non-negligible probability. This concludes the proof of the lemma. □

Lemma 4.8 (Transition from Hybrid H_3 to H_4). *Let OT be equivocal receiver secure, then the hybrids H_3 and H_4 are indistinguishable.*

Proof. We prove this lemma by contradiction, we assume that there exists an adversary $\mathcal{A}$ that distinguishes between the hybrids H_3 and H_4 with non-negligible probability. We use this adversary to construct an adversary $\mathcal{B}$ that breaks the equivocal receiver security of the OT protocol OT with non-negligible probability.

Using the same argument as in the transition from hybrid H_1 to H_2 (Lemma 4.6), also here $\mathcal{B}$ receives as an auxiliary input the values $(\alpha_k^{i,j}, r_k^{i,j})_{j\in[n]\setminus\{i\},k\in[\kappa]}$, $(y_{k,0}^{j,i}, y_{k,1}^{j,i})_{j\in[n]\setminus\{i\},k\in[\kappa]}$ and $(\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l})_{j,l\in[n],j\neq l,k\in[\kappa]}$ for all $i \in M$ which are the values used for the generation of $\mathsf{msg}^1_{\Gamma,\mathcal{A}}$.

Now, we describe the behavior of the adversary $\mathcal{B}$, which behaves on behalf of the honest parties $i \in H$, interacting with the challenger of the equivocal receiver security of the OT protocol OT, for all the honest parties $i \in H$, and the adversary $\mathcal{A}$. The adversary behaves exactly as in the previous hybrid to generate the messages $\mathsf{msg}^2_{\Gamma,\mathcal{H}}, \mathsf{msg}^3_{\Gamma,\mathcal{H}}, \mathsf{msg}^3_{\mathcal{H}}$ as well as the $\{z_i^{\mathsf{part}}\}_{i\in\mathcal{H}}$ part of the message $\mathsf{msg}^2_{\mathcal{H}}$. To generate the remaining values of the OT protocol for all $i \in H$, the adversary $\mathcal{B}$ submits $(v_i[h] \oplus \mathsf{NAND}(v_i[f] \oplus \alpha, v_i[g] \oplus \beta))$ to its underlying challenger. The underlying challenger will then reply with $(\mathsf{ot}_1^{i,t,\alpha,\beta}, \mu^{i,t,\alpha,\beta})$ which the adversary $\mathcal{B}$ uses to compute the remaining messages $\mathsf{msg}^2_{\mathcal{H}}, \mathsf{msg}^3_{\mathcal{H}}$. Here, the message $\mathsf{msg}^3_{\mathcal{H}}$ is, as already mentioned before, computed as in the previous hybrid using the value $\mu^{i,t,\alpha,\beta}$ received from the challenger.

We observe that if the underlying challenger generates the OT message using the honest input $(v_i[h] \oplus \mathsf{NAND}(v_i[f] \oplus \alpha, v_i[g] \oplus \beta))$ on behalf of the honest parties then the hybrid H_3 is simulated towards $\mathcal{A}$ and if the underlying challenger generates the OTs using the equivocator $\mathsf{Sim}^{Eq}_{\mathsf{OT}}$, then the hybrid H_4 is simulated towards $\mathcal{A}$. This directly results in the fact that if $\mathcal{A}$ can distinguish the hybrids H_3 and H_4 with non-negligible probability then $\mathcal{B}$ can also break the equivocal receiver security with non-negligible probability. This concludes the proof of the lemma. □

Lemma 4.9 (Transition from Hybrid H_5 to H_6). *Let* SKE *be a semantic secure symmetric encryption scheme, then the hybrids* H_5 *and* H_6 *are indistinguishable.*

Proof. We prove this lemma by contradiction, we assume that there exists an adversary $\mathcal{A}$ that distinguishes between the hybrids H_5 and H_6 with non-negligible probability. We use this adversary to construct an adversary $\mathcal{B}$ that breaks the semantic security of the symmetric encryption scheme SKE with non-negligible probability. More precisely, here, the adversary $\mathcal{B}$ interacts with multiple instances of a symmetric encryption scheme challenger such that the ciphertexts for all the required instances (for all $i \in H$ and $j, j' \in [n]$ with $j \neq j'$ and $k \in [\kappa]$) can be generated. In this setting, all the challengers use the same challenge bit.

Using the same argument as in the transition from hybrid H_1 to H_2 (Lemma 4.6), also here $\mathcal{B}$ receives as an auxiliary input the values $(\alpha_k^{i,j}, r_k^{i,j})_{j\in[n]\setminus\{i\},k\in[\kappa]}, (y_{k,0}^{j,i}, y_{k,1}^{j,i})_{j\in[n]\setminus\{i\},k\in[\kappa]}$ and $(\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l})_{j,l\in[n],j\neq l,k\in[\kappa]}$ for all $i \in M$ which are the values used for the generation of $\mathsf{msg}^1_{\Gamma,\mathcal{A}}$.

Now, we describe the behavior of the adversary $\mathcal{B}$, which behaves on behalf of the honest parties $i \in H$, interacting with the challenger of the semantic secure symmetric encryption scheme SKE, for all the honest parties $i \in H$ and $j, j' \in [n]$ with $j \neq j'$ and $k \in [\kappa]$, and the adversary $\mathcal{A}$.

The adversary behaves exactly as in the previous hybrid to generate the messages $\mathsf{msg}^2_{\Gamma,\mathcal{H}}, \mathsf{msg}^2_{\mathcal{H}}, \mathsf{msg}^3_{\Gamma,\mathcal{H}}$ as well as the $\{\widetilde{C}^{i,t}\}_{t\in[T]}$ and $\{\mathsf{lab}^{i,1}_{k,\mathsf{st}[k]}\}_{k\notin[(j-1)\ell/n+1,(j-1)\ell/n+(n-1)\kappa]}$ as part of the message $\mathsf{msg}^3_{\mathcal{H}}$ for all $i \in H$. To generate the remaining ciphertexts $(\mathsf{ct}_{k,0}^{i,j,j'}, \mathsf{ct}_{k,1}^{i,j,j'})_{j,j'\in[n],j\neq j',k\in[\kappa]}$ for all $i \in H$, the adversary $\mathcal{B}$ submits $(\mathsf{lab}^{i,1}_{\mathsf{GetIndex}(j,j',k),1-\mathsf{st}^*[\mathsf{GetIndex}(j,j',k)]},$
$\mathsf{lab}^{i,1}_{\mathsf{GetIndex}(j,j',k),\mathsf{st}^*[\mathsf{GetIndex}(j,j',k)]})$ to its underlying challenger for all $i \in H$ and $j, j' \in [n]$ with $j \neq j'$ and $k \in [\kappa]$. The underlying challenger will then reply with $\mathsf{ct}_{k,0}^{i,j,j'}$ for all $i \in H$ and $j, j' \in [n]$ with $j \neq j'$ and $k \in [\kappa]$. To generate the remaining ciphertexts $\mathsf{ct}_{k,1}^{i,j,j'}$, the adversary $\mathcal{B}$ computes $\mathsf{Enc}(\mathsf{sk}_{k,1}^{i,j,j'}, \mathsf{lab}^{i,1}_{\mathsf{GetIndex}(j,j',k),\mathsf{st}^*[\mathsf{GetIndex}(j,j',k)]})$ for all $i \in H$ and $j, j' \in [n]$ with $j \neq j'$ and $k \in [\kappa]$ where $\mathsf{sk}_{k,1}^{i,j,j'}$ are part of the execution of Γ. These ciphertexts are then used to conclude the computation of the message $\mathsf{msg}^3_{\mathcal{H}}$ which is sent to $\mathcal{A}$.

We observe that if the underlying challenger generates the ciphertexts $(\mathsf{ct}_{k,0}^{i,j,j'})_{j,j'\in[n],j\neq j',k\in[\kappa]}$ by encrypting $\mathsf{lab}^{i,1}_{\mathsf{GetIndex}(j,j',k),1-\mathsf{st}^*[\mathsf{GetIndex}(j,j',k)]}$, then the hybrid H_5 is simulated towards $\mathcal{A}$ and if the underlying challenger generates the ciphertext $(\mathsf{ct}_{k,0}^{i,j,j'})_{j,j'\in[n],j\neq j',k\in[\kappa]}$ by encrypting $\mathsf{lab}^{i,1}_{\mathsf{GetIndex}(j,j',k),\mathsf{st}^*[\mathsf{GetIndex}(j,j',k)]}$, then the hybrid H_6 is simulated towards $\mathcal{A}$. This directly results in the fact that if $\mathcal{A}$ can distinguish the hybrids H_5 and H_6 with non-negligible probability then $\mathcal{B}$ can also break the semantic security of the symmetric encryption scheme with non-negligible probability. This concludes the proof of the lemma. □

Lemma 4.10 (Transition from Hybrid H_6 to H_7). *Let GC be a secure garbled circuit, then the hybrids $\mathsf{H}_{7,t-1}$ and $\mathsf{H}_{7,t}$ are indistinguishable for all $t \in [T]$.*

Proof. We prove this lemma by contradiction, we assume that there exists an adversary $\mathcal{A}$ that distinguishes between the hybrids $\mathsf{H}_{7,t-1}$ and $\mathsf{H}_{7,t}$ for any $t \in [T]$ with non-negligible probability. We use this adversary to construct an adversary $\mathcal{B}$ that breaks the security of the garbled circuit GC with non-negligible probability.

Using the same argument as in the transition from hybrid H_1 to H_2 (Lemma 4.6), also here $\mathcal{B}$ receives as an auxiliary input the values $(\alpha_k^{i,j}, r_k^{i,j})_{j\in[n]\setminus\{i\},k\in[\kappa]}$, $(y_{k,0}^{j,i}, y_{k,1}^{j,i})_{j\in[n]\setminus\{i\},k\in[\kappa]}$ and $(\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l})_{j,l\in[n],j\neq l,k\in[\kappa]}$ for all $i \in M$ which are the values used for the generation of $\mathsf{msg}^1_{\Gamma,\mathcal{A}}$.

Now, we describe the behavior of the adversary $\mathcal{B}$, which behaves on behalf of the honest parties $i \in H$, interacting with the challenger of the garbled circuit GC, for all the honest parties $i \in H$, and the adversary $\mathcal{A}$. The adversary behaves exactly as in the previous hybrid to generate the messages $\mathsf{msg}^2_{\Gamma,\mathcal{H}}, \mathsf{msg}^2_{\mathcal{H}}, \mathsf{msg}^3_{\Gamma,\mathcal{H}}$. For the generation of the message $\mathsf{msg}^3_{\mathcal{H}}$, the adversary $\mathcal{B}$ behaves as before with a change in execution of the t'th round of the protocol Φ; namely, the action $\phi_t = (i^*, f, g, h)$. This change works as follows: it completes the execution of Φ using the honest parties inputs and randomness, where the messages of the the corrupted parties are generated via faithful execution. In more detail, st^* is used to start the mental execution of Φ. In this computation phase, the honest parties messages are generated using the inputs and random coins of the honest parties and the messages of each of the malicious parties P_i are generated by executing $\mathsf{Faithful}(i, \mathsf{st}^*, \{b^{i,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta})$. Let $\mathsf{Z} \in \{0,1\}^T$ be the transcript obtained using the above step. Let st^*_T be the public state of one of the corrupted parties a the end of faithful execution and let st^*_t be the public state of the parties at the end of the t'th round of the computation phase. Finally, let $\alpha^* := \mathsf{st}^*_T[f], \beta^* := \mathsf{st}^*_T[g]$ and $\gamma^* := \mathsf{st}^*_T[h]$. To generate the messages for the other honest parties $i \in H \setminus \{i^*\}$, we submit to challenges to the underlying challenger, the first one being $((C^{i,t}[v_i, \bot, \{\mathsf{OT}_2(\mathsf{lab}^{i,t+1}_{b^{i,t,\alpha,\beta}}, \mathsf{lab}^{i,t+1}_{b^{i,t,\alpha,\beta}})\}_{\alpha,\beta}, \mathsf{lab}^{i,t+1}])$, $((\mathsf{OT}_2(\mathsf{ot}_1^{i^*,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}), \{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]\setminus\{h\}}))$ for which the adversary $\mathcal{B}$ will receive as a reply $(\widetilde{C}^{i,t}, \{\mathsf{lab}_k^{i,t}\}_{k\in[\ell]})$ and, if $i^* \in M$, the adversary $\mathcal{B}$ submits the second query $((C^{i,t}[v_i, \{\mu^{i^*,t,\alpha^*,\beta^*}_{v_{i^*}[h]\oplus\mathsf{NAND}(v_{i^*}[f]\oplus\alpha^*,v_{i^*}[g]\oplus\beta^*)}\}_{\alpha,\beta}, \bot, \mathsf{lab}^{i,t+1}])$, $((\alpha^*, \beta^*, \gamma^*), \mu^{i^*,t,\alpha^*,\beta^*}_{\mathsf{Z}_t}, \{\mathsf{lab}^{i^*,t+1}_{k,\mathsf{st}^*_t[k]}\}_{k\in[\ell]}))$ to which it will receive as a reply $(\widetilde{C}^{i^*,t}, \{\mathsf{lab}_k^{i^*,t}\}_{k\in[\ell]})$. These values are then used to generate the final message $\mathsf{msg}^3_{\mathcal{H}}$ in which the ciphertexts $(\mathsf{ct}_{k,0}^{i,j,j'}, \mathsf{ct}_{k,1}^{i,j,j'})_{j,j'\in[n],j\neq j',k\in[\kappa]}$ are generated as in the previous hybrid.

We observe that if the underlying challenger generates the garbled circuits using the submitted circuits, then the hybrid $\mathsf{H}_{7,t-1}$ is simulated towards $\mathcal{A}$ and if the underlying challenger simulates the garbled circuits using the submitted outputs, then the hybrid $\mathsf{H}_{7,t}$ is simulated towards $\mathcal{A}$. This directly results in the fact that if $\mathcal{A}$ can distinguish the hybrids $\mathsf{H}_{7,t-1}$ and $\mathsf{H}_{7,t}$ with non-negligible probability then $\mathcal{B}$ can also break the equivocal receiver security with non-negligible probability. Taking into account that the above argument holds for all $t \in [T]$ implies that $\mathsf{H}_6 = \mathsf{H}_{7,0}$ and $\mathsf{H}_7 = \mathsf{H}_{7,T}$ are indistinguishable. □

Lemma 4.11 (Transition from Hybrid H_8 to H_9). *Let Φ be statistical secure, then the hybrids H_8 and H_9 are indistinguishable.*

Proof. We prove this lemma by contradiction, we assume that there exists an adversary $\mathcal{A}$ that distinguishes between the hybrids H_8 and H_9 with non-negligible probability. We use this adversary to construct an adversary $\mathcal{B}$ that breaks the security of the protocol Φ with non-negligible probability.

Using the same argument as in the transition from hybrid H_1 to H_2 (Lemma 4.6), also here $\mathcal{B}$ receives as an auxiliary input the values $(\alpha_k^{i,j}, r_k^{i,j})_{j\in[n]\setminus\{i\},k\in[\kappa]}$, $(y_{k,0}^{j,i}, y_{k,1}^{j,i})_{j\in[n]\setminus\{i\},k\in[\kappa]}$ and $(\mathsf{sk}_{k,0}^{i,j,l}, \mathsf{sk}_{k,1}^{i,j,l})_{j,l\in[n],j\neq l,k\in[\kappa]}$ for all $i \in M$ which are the values used for the generation of $\mathsf{msg}^1_{\Gamma,\mathcal{A}}$.

Now, we describe the behavior of the adversary $\mathcal{B}$, which behaves on behalf of the honest parties $i \in H$, interacting with the challenger of the security of the protocol Φ, for all the honest parties $i \in H$, and the adversary $\mathcal{A}$. The adversary behaves exactly as in the previous hybrid to generate the messages $\mathsf{msg}^2_{\Gamma,\mathcal{H}}, \mathsf{msg}^2_{\mathcal{H}}, \mathsf{msg}^3_{\Gamma,\mathcal{H}}$. To generate the remaining message $\mathsf{msg}^3_{\mathcal{H}}$, the adversary $\mathcal{B}$ interacts with its underlying challenger. In more detail, the adversary $\mathcal{B}$ generates the tuple $(H, \mathsf{st}^*, \mathsf{aux})$ as described in the simulator and then submits $((\{b^{i,t\alpha,\beta}\}_{t\in A_i,\alpha,\beta})_{i\in H}, (H, \mathsf{st}^*, \mathsf{aux}))$ to the underlying challenger, where the values $(\{b^{i,t\alpha,\beta}\}_{t\in A_i,\alpha,\beta})_{i\in H}$ are part of the inputs of the honest parties. To generate the corresponding messages of the malicious parties, the adversary $\mathcal{B}$ executes $\mathsf{Faithful}(i, \mathsf{st}^*, \{b^{i,t\alpha,\beta}\}_{t\in A_i,\alpha,\beta})$ for all $i \in M$. and uses the output of this procedure in the interaction with the challenger. Once this interaction completes, the adversary $\mathcal{B}$ can use the obtained values of the protocol Φ to generate the final message $\mathsf{msg}^3_{\mathcal{H}}$ as described in the previous hybrid.

We observe that if the underlying challenger generates the messages of Φ using the honest inputs $(\{b^{i,t\alpha,\beta}\}_{t\in A_i,\alpha,\beta})_{i\in H}$, then the hybrid H_8 is simulated towards $\mathcal{A}$ and if the underlying challenger simulates the messages of Φ, then the hybrid H_9 is simulated towards $\mathcal{A}$. This directly results in the fact that if $\mathcal{A}$ can distinguish the hybrids H_8 and H_9 with non-negligible probability then $\mathcal{B}$ can also break the equivocal receiver security with non-negligible probability. This concludes the proof of the lemma. □

5 Our Four-Round MPC Protocol

Our four-round black-box MPC protocol makes use of the following primitives.

1. A MAC scheme MAC.

2. The Outer Protocol $\Phi = (\Phi_1, \Phi_2)$, a 2-round, n-client, m-server MPC protocol achieving privacy with knowledge of outputs (Remark 2.8) against a malicious, adaptive adversary corrupting up to $n-1$ clients and $t = (m-1)/3$ servers. Φ realizes the functionality g, wich takes the inputs $(z_1, \ldots, z_n)$, and does the following:
   (a) For each $i \in [n]$ parse z_i as x_i, k_i.
   (b) Compute $y := f(x_1, \ldots, x_n)$.
   (c) Compute $\sigma_i := \mathsf{MAC}(\mathsf{k}_i, y)$ for all $i \in [n]$
   (d) Return $(y, \sigma_1, \ldots, \sigma_n)$.

   In our protocol, we set $t = 2\lambda n^2$ and refer to Section 2.6 for more details about the outer protocol we use. All the properties required by our outer protocol are satisfied by the construction proposed in [IKP10, IKSS21].
3. A simultaneous OT protocol that satisfies Definitions 2.10 and 2.11 $\mathsf{WL} = (\mathsf{WL}_1, \mathsf{WL}_2, \mathsf{WL}_3)$ for the functionality $\mathcal{F}^{\mathsf{list}}_{OT^{\lambda,m}}$ that is secure against sometimes aborting adversaries. The functionality is parametrized by $\mathcal{X}_i$, which represents the input space of the senders $S_{j,i}$ for each $j \in [n]$. The sampler of $\mathcal{X}_i$ is defined by the following process

   Initialize the empty list Y. For each $h \in [m]$:
   (a) Sample a set of random seeds s_i, and let S_i be the evaluation of a PRG on each of the sampled seeds, also sample $r_{i,h}, r^{\mathsf{prg}}_{i,h} \leftarrow \{0,1\}^\lambda$.
   (b) Add $(r_{i,h}, r^{\mathsf{prg}}_{i,h}, s_{i,h}, S_{i,h})$ to Y.

   return $Z = \{Y, \ldots, Y\}$ with $|Z| = n$.

   We refer to this protocol as the Watchlist Protocol.
4. A $2n$-party protocol $\Pi = (\{\Pi^{\mathsf{no\text{-}inp}}_i\}_{i\in[3]}, \{\Pi^{\mathsf{inp}}_i\}_{i\in\{1,2\}}, \Pi^{\mathsf{out}})$ that satisfies Definition 4.1 and realizes the function Φ_2. The function Φ_2 has n inputs (one for each of the clients of the outer protocol), and each input is a tuple (s_i, S_i, ϕ_i), where, as discussed in Section 2.6, s_i denotes a set of seeds sampled by the clients of the outer protocol Φ, S_i contains the PRG evaluations of the seeds, and ϕ_i denotes the first round of the client of the original protocol described in [IKP10]. In our final MPC protocol we will have n parties, each running two sub-parties in each execution of the *inner protocol* Π. The first sub-party will use as input the sets (s_i, S_i), and the other sub-party will use the ϕ_i. We denote the maximum size (in bits) of a message of the protocol with ℓ.
5. The nRmS OT $\mathsf{OT}_{\vec{S},\vec{R}} := \{(\mathsf{OT}^i_{R,1}, \mathsf{OT}^i_{S,2}, \mathsf{OT}^i_{R,3}, \mathsf{OT}^i_{S,4}, \mathsf{OT}^i_{\mathsf{out}})\}_{i\in n}$ of Section 3 that realizes the functionality $\mathcal{F}_{n,n-1}$ (see Figure 3.3) accordingly to Definition 3.6. More precisely, we have n receivers and each receiver interacts with $(n-1)$ senders. In particular, the input of each receiver corresponds to the concatenation of all the second-round messages of Π generated by the party using the algorithm Π^{inp}_1 (i.e., the input of the receiver is a bit string of total size $m\ell$). The input of the sender $S_{i,j}$ corresponds to the labels of the garbled circuit described in Figure 5.2 needed to encode the i-th input (represented by a message of Π generated using Π^{inp}_1) to the circuit.
6. A garbling scheme $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Eval})$. We denote with $\mathsf{Sim}_{\mathsf{GC}}$ the simulator of the scheme. As anticipated, the circuit that will be garbled is described in Figure 5.2.

We propose the formal description of our protocol in Figure 5.1 and refer to reader to the introductory section for a high-level overview of the scheme.

Figure 5.1: Our MPC protocol

Each party P_i has an input x_i, and in act as described below (with $i \in [n]$).

Round 1.

1. Choose a random MAC key $\mathsf{k}_i \leftarrow \{0,1\}^*$ and set $z_i := (x_i, \mathsf{k}_i)$.
2. Compute $(\phi_1^{i \to 1}, \ldots, \phi_1^{i \to m}) \leftarrow \Phi_1(1^\lambda, i, z_i)$.
3. Sample a random subset $K_I \subset [m]$ of size λ and set $x_{i,j} := K_i$ for all $j \in [n] \setminus \{i\}$.
4. Sample the PRG seeds and evaluate the PRG over these seeds. We denote with $s_{i,h}$ the set containing all the seeds that will be sent to the h-th server and with $S_{i,h}$ the set containing the evaluations of the PRG on the seeds contained in $s_{i,h}$, with $h \in [m]$.
5. Sample $r_{i,h}, r_{i,h}^{\mathsf{prg}} \leftarrow \{0,1\}^\lambda$ for all $h \in [m]$ and set $y_{i,j} := \{r_{i,h}, r_{i,h}^{\mathsf{prg}}, s_{i,h}, S_{i,h}\}_{h \in [m]}$ for all $j \in [n] \setminus \{i\}$.
6. Compute $\mathsf{wl}_1^i \leftarrow \mathsf{WL}_1(1^\lambda, i, \{x_{i,j}, y_{i,j}\}_{j \in [n] \setminus \{i\}}, \mathsf{wl}(0))$ (here $\mathsf{wl}(r)$ denotes the transcript in the first r rounds of WL).
7. Sample $\rho^i \leftarrow \{0,1\}^\lambda$ and compute $\mathsf{ot}_1^i \leftarrow \mathsf{OT}_{R,1}^i(1^\lambda; \rho^i)$.
8. Broadcast $(\mathsf{wl}_1^i, \mathsf{ot}_1^i)$.

Round 2.

1. Compute $\mathsf{wl}_2^i \leftarrow \mathsf{WL}_2(1^\lambda, i, \{x_{i,j}, y_{i,j}\}_{j \in [n] \setminus \{i\}}, \mathsf{wl}(1))$.
2. $\pi_{h,1}^i := \Pi_{h,1}^{\mathsf{no\text{-}inp}}(1^\lambda, i; r_{i,h})$, $\pi_{h,1}^{\mathsf{prg},i} := \Pi_{h,1}^{\mathsf{no\text{-}inp}}(1^\lambda, i; r_{i,h}^{\mathsf{prg}})$ for all $h \in [m]$.
3. For each $j \in [n] \setminus \{i\}$ compute $\mathsf{ot}_2^{i,j} \leftarrow \mathsf{OT}_{S,2}^j(\mathsf{ot}(1))$ (here $\mathsf{ot}(r)$ denotes the transcript in the first r rounds of $\mathsf{OT}_{\vec{S},\vec{R}}$).
4. Broadcast $\mathsf{wl}_2^i, \{\pi_{h,1}^i, \pi_{h,1}^{\mathsf{prg},i}\}_{h \in [m]}, \{\mathsf{ot}_2^{i,j}\}_{j \in [n] \setminus \{i\}}$.

Round 3.

1. Compute $\mathsf{wl}_3^i \leftarrow \mathsf{WL}_3(1^\lambda, i, \{x_{i,j}, y_{i,j}\}_{j \in [n] \setminus \{i\}}, \mathsf{wl}(2))$.
2. For each $h \in [m]$ compute the following
   - $(z_h, \pi_{h,2}^i) \leftarrow \Pi_2^{\mathsf{no\text{-}inp}}(i, 1^\lambda, \pi_h(1); r_{i,h})$, $\tilde{\pi}_{h,1}^i \leftarrow \Pi_1^{\mathsf{inp}}(i, \phi_1^{i \to h}, z_h)$
   - $(z_h^{\mathsf{prg}}, \pi_{h,2}^{i,\mathsf{prg}}) \leftarrow \Pi_2^{\mathsf{no\text{-}inp}}(i, 1^\lambda, \pi_h^{\mathsf{prg}}(1); r_{i,h}^{\mathsf{prg}})$, $\tilde{\pi}_{h,1}^{i,\mathsf{prg}} \leftarrow \Pi_1^{\mathsf{inp}}(i, (s_{i,h}, S_{i,h}), z_h^{\mathsf{prg}})$
3. Set $\mathsf{otinp}^i := (\tilde{\pi}_{h,1}^i)_{h \in [m]}$, compute $\mathsf{ot}_3^i \leftarrow \mathsf{OT}_{R,3}^i(\mathsf{otinp}^i, \mathsf{ot}^i(2); \rho^i)$.
4. Broadcast $\mathsf{wl}_3^i, \{\pi_{h,2}^i, \pi_{h,2}^{\mathsf{prg},i}, \tilde{\pi}_{h,1}^{i,\mathsf{prg}}\}_{h \in [m]}, \mathsf{ot}_3^i$.

Round 4.

1. Run $\mathsf{out}_{\mathsf{WL}}$ on input the randomness used to compute WL messages and $\mathsf{wl}(4)$ thus obtaining $\{r_{j,h}, r_{j,h}^{\mathsf{prg}}, (s_{j,h}, S_{j,h})\}_{j \in [n] \setminus \{i\}, h \in K_i}$.
2. For all $j \in [n] \setminus \{i\}$ and $h \in K_i$, check that:
   - The PRG computations in $(s_{j,h}, S_{j,h})$ are correct.
   - Compute $\pi_{h,1}^{\star j} := \Pi_{h,1}^{\mathsf{no\text{-}inp}}(1^\lambda, j, \pi_h(0); r_{j,h})$ and $\pi_{h,1}^{\star j,\mathsf{prg}} := \Pi_{h,1}^{\mathsf{no\text{-}inp}}(1^\lambda, j, \pi_h(0); r_{j,h}^{\mathsf{prg}})$, check that $\pi_{h,1}^{\star j} = \pi_{h,1}^j$ and $\pi_{h,1}^{\star j,\mathsf{prg}} = \pi_{h,1}^{j,\mathsf{prg}}$.
   - Compute $(z_h, \pi_{h,2}^{j\star}) := \Pi_2^{\mathsf{no\text{-}inp}}(j, 1^\lambda, \pi_h(1); r_{j,h})$ check that $\pi_{h,2}^{j\star} = \pi_{h,2}^j$.
   - Compute $(z_h^{\mathsf{prg}}, \pi_{h,2}^{\mathsf{prg},j\star}) := \Pi_2^{\mathsf{no\text{-}inp}}(j, 1^\lambda, \pi_h(1), r_{j,h}^{\mathsf{prg}})$ and $\tilde{\pi}_{1,h}^{\mathsf{prg},j\star} := \Pi_1^{\mathsf{inp}}(j, (s_{j,h}, S_{j,h}), z_h^{\mathsf{prg}})$, check that $\pi_{h,2}^{\mathsf{prg},j\star} = \pi_{h,2}^{\mathsf{prg},j}$ and $\tilde{\pi}_{1,h}^{\mathsf{prg},j\star} = \tilde{\pi}_{1,h}^{\mathsf{prg},j}$
3. If any of the above checks fail, output $\perp$.

4. For each $h \in [m]$ compute $\tilde{\mathsf{C}}_h, \{\mathsf{lab}_0^{j,k,h}, \mathsf{lab}_1^{j,k,h}\}_{j\in[n]\setminus\{i\},k\in[\ell],h\in[m]} \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{C}_h[\phi_1^{i\to h}, s_{i,h}, S_{i,h}, z_h, z_h^{\mathsf{prg}}, \{\tilde{\pi}_{h,1}^{\mathsf{prg},j}\}_{j\in[n]}])$
5. For each $j \in [n] \setminus \{i\}$, compute $\mathsf{ot}_4^{j,i} \leftarrow \mathsf{OT}_{S,4}^j(1^\lambda, ((\mathsf{lab}_0^{j,k,h}, \mathsf{lab}_1^{j,k,h})_{k\in[\ell]})_{h\in[m]}, \mathsf{ot}(2))$.
6. For all $h \in [m]$, compute $\pi_{h,3}^i := \Pi_{h,3}^{\mathsf{no\text{-}inp}}(1^\lambda, i, \pi_h(2); r_{i,h})$ $\pi_{h,3}^{\mathsf{prg},i} := \Pi_{h,3}^{\mathsf{no\text{-}inp}}(1^\lambda, i, \pi_h(2); r_{i,h}^{\mathsf{prg}})$.
7. Broadcast $\{\pi_{h,3}^i, \pi_{h,3}^{\mathsf{prg},i}, \tilde{\mathsf{C}}_h\}_{h\in[m]}, \{\mathsf{ot}_4^{j,i}\}_{j\in[n]\setminus\{i\}}, (\mathsf{otinp}^i, \rho^i)$.

Output Computation.

1. For each $j \in [n]$ use the input randomness pair $(\mathsf{otinp}^i, \rho^i)$ to execute $\mathsf{OT}_{\mathsf{out}}^j$ thus obtaining the output of the j-th receiver $((\mathsf{lab}^{j,h,c})_{h\in[m]})_{c\in[n]\setminus\{j\}}$.
2. Let $\{\tilde{\mathsf{C}}_{c,h}\}$ be the set of GCs received from the c-th party in the fourth round. For each $h \in [m], c \in [n]$, $(\tilde{\pi}_{2,h}^c, \tilde{\pi}_{2,h}^{\mathsf{prg},c}) := \mathsf{Eval}(\tilde{\mathsf{C}}_{c,h}, (\mathsf{lab}^{j,h,c})_{j\in[n]\setminus\{i\}})$.
3. For each $h \in [m]$ compute $\phi_2^h \leftarrow \Pi^{\mathsf{out}}(\phi_1^{i\to h}, \pi(2))$
4. Compute $(y, \sigma_1, \dots, \sigma_n) := \mathsf{out}_\Phi(\{\phi_2^h\}_{h\in[m]})$.
5. Check if σ_i is a valid tag on y using the key k_i. If the check is successful output y, otherwise output $\perp$.

Figure 5.2: Circuit C computed by $\mathsf{C}_h[\phi, s, S, z, z^{\mathsf{prg}}, \{\tilde{\pi}_{h,1}^{\mathsf{prg},j}\}_{j\in[n]}]$

On input $\{\tilde{\pi}_{h,1}^j\}_{j\in[n]\setminus\{i\}}$, set $\tilde{\pi} = \{\tilde{\pi}_{h,1}^j, \tilde{\pi}_{h,1}^{\mathsf{prg}}, \}_{j\in[n]\setminus\{i\}}$ compute $\tilde{\pi}_{h,2}^i \leftarrow \Pi_2^{\mathsf{inp}}(i, \phi, z, \tilde{\pi})$, $\tilde{\pi}_2^{\mathsf{prg},i} \leftarrow \Pi_2^{\mathsf{inp}}(i, (s, S), z^{\mathsf{prg}}, \tilde{\pi})$. Return $(\tilde{\pi}_2^i, \tilde{\pi}_2^{\mathsf{prg},i})$

Theorem 5.1. *Assuming the security of the primitives listed above, the protocol of Figure 5.1 is a secure 4-round MPC protocol.*

Simulator. Let $\mathcal{A}$ be an adversary that corrupts the set of parties indexed by M and let $H := [n] \setminus M$. Let $\mathsf{Sim}_{\mathsf{ot}}$ be the simulator of the nRmS OT protocol $\mathsf{OT}_{\vec{S},\vec{R}}$. Let $(\mathsf{Sim}_{\mathsf{WL}}^1, \mathsf{Sim}_{\mathsf{WL}}^2)$ be simulator of the watchlist protocol $\mathsf{WL} = (\mathsf{WL}_1, \mathsf{WL}_2, \mathsf{WL}_3)$. The simulator of Π_{mpc} then works as described in Figure 5.3:

Figure 5.3: Simulator Sim

1. Run $\mathsf{Sim}_{\mathsf{WL}}^1$ against the adversary $\mathcal{M}$ (defined in Figure 5.4) by sending messages on its right interface with the following modifications. When $\mathcal{M}$ waits for the messages of $\mathsf{OT}_{\vec{S},\vec{R}}$ computed on behalf of the honest senders, Sim computes these messages as the honest senders of $\mathsf{OT}_{\vec{S},\vec{R}}$ (note that to compute these messages not input is required), thus obtaining $\{\mathsf{ot}_2^{i,j}\}_{i\in H,j\in M}$ and send them to the right interface of $\mathsf{OT}_{\vec{S},\vec{R}}$ along with the messages of $\mathsf{Sim}_{\mathsf{WL}}^1$.
2. If $\mathsf{Sim}_{\mathsf{WL}}^1$ stops and returns $\perp$, then instruct the ideal functionality to abort and return the view of $\mathcal{M}$. Else, when $\mathsf{Sim}_{\mathsf{WL}}^1$ returns the output $(\{x_{i,j}\}_{i\in M,j\in H}, 1^k)$, emulate the behavior of $\mathcal{F}_{OT^{\lambda,m}}^{\mathsf{list}}$ on input $\{x_{i,j}\}_{i\in M,j\in H}$ to obtain $\{\mathsf{out}_{i,j}^\kappa\}_{i\in H,j\in M,\kappa\in[k]}$.

3. Set $C \coloneqq \{x_{i,j}\}_{i \in M, j \in H}$ and send $(\mathsf{continue}, C)$ to the right interface of $\mathcal{M}$.
4. Run $\mathsf{Sim}_{\mathsf{ot}}$ against the adversary $\mathcal{M}$, and any time that $\mathcal{M}$ needs a message of the watchlist protocol compute it accordingly to $\mathsf{Sim}^1_{\mathsf{WL}}$. If $\mathsf{Sim}_{\mathsf{ot}}$ returns $\perp$, then instruct the ideal functionality to abort. Else, if $\mathsf{Sim}_{\mathsf{ot}}$ returns $\{\mathsf{otinp}^i\}_{i \in M}$ send $\{\mathsf{otinp}^i\}_{i \in M}$ to the right interface of $\mathcal{M}$ along with any information that will be sent in the third round directed to the right interface of $\mathcal{M}$ from now on.
5. Run $\mathsf{Sim}^2_{\mathsf{WL}}(\{\mathsf{out}^{\kappa}_{i,j}\}_{i \in M, j \in H, \kappa \in [k]}, z)$ against the adversary $\mathcal{M}$, emulating $\mathcal{F}^{\mathsf{list}}_{OT^{\lambda,m}}$ with the following modification. When $\mathcal{M}$ waits for the messages of $\mathsf{OT}_{\vec{S},\vec{R}}$ computed on behalf of the honest senders during the second round, compute these messages as the honest senders of $\mathsf{OT}_{\vec{S},\vec{R}}$ would using default inputs, thus obtaining $\{\mathsf{ot}^{i,j}_2\}_{i \in H, j \in M}$ and send them to the right interface of $\mathsf{OT}_{\vec{S},\vec{R}}$ along with the messages of $\mathsf{Sim}^1_{\mathsf{WL}}$.
6. Let $(\iota, \{y_{i,j}\}_{i \in M, j \in H}, \mathrm{View})$ be the output of $\mathsf{Sim}^2_{\mathsf{WL}}$, send $\{y_{i,j}\}_{i \in M, j \in H}$ to the right interface of $\mathcal{M}$ along with the third round of $\mathsf{Sim}^2_{\mathsf{WL}}$ generated in the output thread whenever $\mathcal{M}$ is waiting to receive a message on the right interface after having completed the third round.
7. Upon receiving $\{\mathsf{lab}^{i,h,c}\}_{i \in H, h \in [m], c \in [n] \setminus \{i\}}$ from $\mathcal{M}$, complete the execution of $\mathsf{Sim}_{\mathsf{ot}}$, by relying on its query with the messages $\{\mathsf{lab}^{i,h,c}\}_{i \in H, h \in [m], c \in [n] \setminus \{i\}}$ (the simulated garbled circuit labels computed by $\mathcal{M}$ in the last step) thus obtaining $\{\mathsf{ot}^{i,j}_4\}_{i \in H, j \in M}$, and send it to the right interface of $\mathcal{M}$.
8. Return what $\mathcal{M}$ returns.

Figure 5.4: $\mathcal{M}$

Let Sim_{Π_h} denote the simulator for the h-th inner-protocol, and let Sim_{Φ} denote the simulator for the outer protocol $\Phi = (\Phi_1, \Phi_2, \mathsf{out}_{\Phi})$. This machine has oracle access to $\mathcal{A}$, the adversary attacking our MPC protocol.

Round 1.

1. Upon receiving $\{\mathsf{wl}^i_1\}_{i \in H}$ from the right interface, for each $i \in H$ sample $\rho^i \leftarrow \{0,1\}^{\lambda}$ and compute $\mathsf{ot}^i_1 \leftarrow \mathsf{OT}^i_{R,1}(1^{\lambda}; \rho^i)$ and send $(\mathsf{wl}^i_1, \mathsf{ot}^i_1)$ to $\mathcal{A}$.
2. Upon receiving $(\mathsf{wl}^i_1, \mathsf{ot}^i_1)_{i \in M}$ from $\mathcal{A}$ forward these messages on to the right interface.

Round 2 **(pre-extraction).** Upon receiving $\{(\mathsf{wl}^i_2, y_{i,j}), \mathsf{ot}^{i,j}_2\}_{i \in H, j \in M}$ from the right interface, for each $i \in H, j \in M$ parse $y_{i,j}$ as $\{r_{i,h}, r^{\mathsf{prg}}_{i,h}, s_{i,h}, S_{i,h}\}_{h \in [m]}$ and do the following.

1. For each $h \in [m]$ compute $\pi^i_{h,1} \coloneqq \Pi^{\mathsf{no\text{-}inp}}_{h,1}(1^{\lambda}, i; r_{i,h})$, $\pi^{\mathsf{prg},i}_{h,1} \coloneqq \Pi^{\mathsf{no\text{-}inp}}_{h,1}(1^{\lambda}, i; r^{\mathsf{prg}}_{i,h})$ for all $h \in [m]$.
2. For each $h \in [m]$ send $\mathsf{wl}^i_2, \{\pi^i_{h,1}, \pi^{\mathsf{prg},i}_{h,1}\}_{h \in [m]}, \{\mathsf{ot}^{i,j}_2\}_{j \in [n] \setminus \{i\}}$ to $\mathcal{A}$

Round 3 **(pre-extraction).** Upon receiving $\{\mathsf{wl}^i_2, \{\pi^i_{h,1}, \pi^{\mathsf{prg},i}_{h,1}\}_{h \in [m]}, \{\mathsf{ot}^{i,j}_2\}_{j \in H}\}_{i \in H}$ from $\mathcal{A}$ do the following

1. Forward $\{\mathsf{wl}^i_2\}_{i \in M}$ to the right interface. Upon receiving $\{\mathsf{wl}^i_3\}_{i \in H}$ from the right interface, for each $i \in H$ do as follows.

2. Sample $\mathsf{k}_i \leftarrow \{0,1\}^*$, set $z_i := (0^\lambda, \mathsf{k}_i)$, compute $(\phi_1^{i\to 1}, \ldots, \phi_1^{i\to m}) \leftarrow \Phi_1(1^\lambda, i, z_i)$.
3. For each $h \in [m]$ do the following
   (a) $(z_h, \pi_{h,2}^i) \leftarrow \Pi_2^{\mathsf{no\text{-}inp}}(i, 1^\lambda, \pi_h(1); r_{i,h})$, $\tilde{\pi}_{h,1}^i \leftarrow \Pi_1^{\mathsf{inp}}(i, \phi_1^{i\to 1}, z_h)$
   (b) $(z_h^{\mathsf{prg}}, \pi_{h,2}^{i,\mathsf{prg}}) \leftarrow \Pi_2^{\mathsf{no\text{-}inp}}(i, 1^\lambda, \pi_h^{\mathsf{prg}}(1); r_{i,h}^{\mathsf{prg}})$, $\tilde{\pi}_{h,1}^{i,\mathsf{prg}} \leftarrow \Pi_1^{\mathsf{inp}}(i, (s_{i,h}, S_{i,h}), z_h^{\mathsf{prg}})$
4. Set $\mathsf{otinp}^i := (\tilde{\pi}_{h,1}^i)_{h\in[m]}$, compute $\mathsf{ot}_3^i \leftarrow \mathsf{OT}_{R,3}^i(\mathsf{otinp}^i, \mathsf{ot}^i(2); \rho^i)$.
5. Send $\mathsf{wl}_3^i, \{\pi_{h,2}^i, \pi_{h,2}^{\mathsf{prg},i}, \tilde{\pi}_{h,1}^{i,\mathsf{prg}}\}_{h\in[m]}, \mathsf{ot}_3^i$ to $\mathcal{A}$.

Extraction phase. Upon receiving $\{\mathsf{wl}_3^i, \{\pi_{h,2}^i, \pi_{h,2}^{\mathsf{prg},i}, \tilde{\pi}_{h,1}^{i,\mathsf{prg}}\}_{h\in[m]}, \mathsf{ot}_3^i\}_{i\in M}$ from $\mathcal{A}$, forward $\{\mathsf{wl}_3^i, \mathsf{ot}_3^i\}_{i\in M}$ to the right interface. If the right interface sends the message abort, then return the view of $\mathcal{A}$ and stop, else, if the right interface sends $(\mathsf{continue}, C)$ change the way in which the second and the third round of the protocol are computed as described below.

Post-extraction 2-nd round. Upon receiving $\{\mathsf{wl}_2^i, y_i, \mathsf{ot}_2^{i,j}\}_{i\in H, j\in M}$ from the right interface, for each $i \in H, j \in M$ parse $y_{i,j}$ as $\{r_{i,h}, r_{i,h}^{\mathsf{prg}}, s_{i,h}, S_{i,h}\}_{h\in C}$ and do the following.

1. For each $h \notin C$, compute $\{\pi_{h,1}^i, \pi_{h,1}^{\mathsf{prg},i}\}_{i\in[n]\setminus C} \leftarrow \mathsf{Sim}_{\Pi_h}(1^\lambda)$ (where Sim_{Π_h} is executed with a set of corrupted parties C).
2. For each $h \in C$ compute $\pi_{h,1}^i := \Pi_{h,1}^{\mathsf{no\text{-}inp}}(1^\lambda, i; r_{i,h})$, $\pi_{h,1}^{\mathsf{prg},i} := \Pi_{h,1}^{\mathsf{no\text{-}inp}}(1^\lambda, i; r_{i,h}^{\mathsf{prg}})$ for all $h \in [m]$.
3. For each $i \in H$ send $\mathsf{wl}_2^i, \{\pi_{h,1}^i, \pi_{h,1}^{\mathsf{prg},i}\}_{h\in[m]}, \{\mathsf{ot}_2^{i,j}\}_{j\in M}$ to $\mathcal{A}$

Post-extraction 3rd round. Upon receiving $\{\mathsf{wl}_2^i, \{\pi_{h,1}^i, \pi_{h,1}^{\mathsf{prg},i}\}_{h\in[m]}, \{\mathsf{ot}_2^{i,j}\}_{j\in H}\}_{i\in H}$ from $\mathcal{A}$ do the following steps.

1. Forward $\{\mathsf{wl}_2^i\}_{i\in M}$ to the right interface. Upon receiving $\{\mathsf{wl}_3^i\}_{i\in H}$ from the right interface, and for each $i \in H$ do as follows.
2. Invoke Sim_Φ by corrupting the set of clients indexed by M and corrupting the set of servers indexed by C, thus obtaining $\{\phi_1^{i\to j}\}_{i\in H, j\in C}$.
3. For each $h \notin C$ compute $\{(\pi_{h,2}^i, \tilde{\pi}_{h,1}^i), (\pi_{h,2}^{i,\mathsf{prg}}, \tilde{\pi}_{h,1}^{i,\mathsf{prg}})\}_{i\in H} \leftarrow \mathsf{Sim}_{\Pi_h}(\pi(1))$
4. For each $h \in C, i \in H$ compute $(z_h, \pi_{h,2}^i) \leftarrow \Pi_2^{\mathsf{no\text{-}inp}}(i, 1^\lambda, \pi_h(1); r_{i,h})$, $\tilde{\pi}_{h,1}^i \leftarrow \Pi_1^{\mathsf{inp}}(i, \phi_1^{i\to h}, z_h)$ and $(z_h^{\mathsf{prg}}, \pi_{h,2}^{i,\mathsf{prg}}) \leftarrow \Pi_2^{\mathsf{no\text{-}inp}}(i, 1^\lambda, \pi_h^{\mathsf{prg}}(1); r_{i,h}^{\mathsf{prg}})$, $\tilde{\pi}_{h,1}^{i,\mathsf{prg}} \leftarrow \Pi_1^{\mathsf{inp}}(i, (s_{i,h}, S_{i,h}), z_h^{\mathsf{prg}})$
5. Set $\mathsf{otinp}^i := (\tilde{\pi}_{h,1}^i)_{h\in[m]}$, compute $\mathsf{ot}_3^i \leftarrow \mathsf{OT}_{R,3}^i(\mathsf{otinp}^i, \mathsf{ot}^i(2); \rho^i)$.
6. Send $\mathsf{wl}_3^i, \{\pi_{h,2}^i, \pi_{h,2}^{\mathsf{prg},i}, \tilde{\pi}_{h,1}^{i,\mathsf{prg}}\}_{h\in[m]}, \mathsf{ot}_3^i$ to $\mathcal{A}$.

Round 4. Upon receiving $\{\mathsf{wl}_3^i, \{\pi_{h,2}^i, \pi_{h,2}^{\mathsf{prg},i}, \tilde{\pi}_{h,1}^{i,\mathsf{prg}}\}_{h\in[m]}, \mathsf{ot}_3^i\}_{i\in M}$ from $\mathcal{A}$ do as follows.

1. Forward $\{\mathsf{wl}_3^i, \mathsf{ot}_3^i\}_{i\in M}$ to the right interface. And upon receiving $(\{y_{i,j}\}_{i\in M, j\in H}, \{\mathsf{otinp}^i\}_{i\in M})$ from the right interface (where $\{y_{i,j}\}_{i\in M, j\in H}$ represent the values extracted related to the watchlist senders' inputs, and $\{\mathsf{otinp}^i\}_{i\in M}$ represent the receivers' input used by the adversary in the execution of nRmS), parse otinp^i as $(\tilde{\pi}_{h,1}^i)_{h\in[m]}$ (with $i \in M$) and continue as follows.
2. Initialize the empty set C'. For each $h \in [m]$, check if there exists some $j \in H$ such that for every $i \in M, y_{i,j}$ contains the randomness that explains the messages sent by corrupted parties in Π_h as well as the correct PRG computations. If not, it adds h to C'.

3. If $|C'| > \lambda n^2$ stop and and instructs all the honest parties to output $\perp$, else continue.
4. Instruct Sim_Φ to additionally corrupt the servers indexed by C' thus obtaining $\{\phi_1^{i \to j}\}_{i \in H, j \in C'}$.
5. For each $h \in C, i \in H$
   (a) Set $\tilde{\pi} = \{\tilde{\pi}_{h,1}^j, \tilde{\pi}_{h,1}^{\mathsf{prg}}, \}_{j \in [n] \setminus \{i\}}$
   (b) Compute $\pi_{h,3}^i := \Pi_{h,3}^{\mathsf{no\text{-}inp}}(1^\lambda, i, \pi_h(2); r_{i,h})$ $\pi_{h,3}^{\mathsf{prg},i} := \Pi_{h,3}^{\mathsf{no\text{-}inp}}(1^\lambda, i, \pi_h(2); r_{i,h}^{\mathsf{prg}})$ compute $\tilde{\pi}_{h,2}^i \leftarrow \Pi_2^{\mathsf{inp}}(i, \phi_1^{i \to h}, z_{i,h}, \tilde{\pi})$, $\tilde{\pi}_2^{\mathsf{prg},i} \leftarrow \Pi_2^{\mathsf{inp}}(i, (s_{i,h}, S_{i,h}), z_{i,h}^{\mathsf{prg}}, \tilde{\pi})$.
6. For each $h \in C'$
   (a) $\{\pi_{h,3}^i, \tilde{\pi}_{h,2}^i, \pi_{h,3}^{\mathsf{prg},i}, \tilde{\pi}_{h,2}^{\mathsf{prg},i}\}_{i \in H} \leftarrow \mathsf{Sim}_{\Pi_h}(1^\lambda, i, (\{\phi_1^{i \to h}, s_{i,h}, S_{i,h})\}_{i \in H})$
7. For each $h \in [m] \setminus \{C' \cup C\}$ run $\mathsf{Sim}_{\Pi_h}(1^\lambda, i, \{r_{j,h}, r_{j,h}^{\mathsf{prg}}\}_{j \in M})$ and wait for its query.
8. Let $\{\phi_{h,1}^i\}_{i \in M}$ be the query made by the h-th simulator Sim_{Π_h}, with $h \in [m] \setminus \{C' \cup C\}$, send $\{\phi_1^{i \to h}\}_{i \in M, h \in [m] \setminus \{C \cup C'\}}$ to Sim_Φ. When Sim_Φ queries its ideal functionality g with the inptus $\{z_i = (x_i, \mathsf{k}_i)\}_{i \in M}$ send $\{x_i\}_{i \in M}$ to the ideal functionality f thus obtaining y. For each $i \in [n]$ compute $\sigma_i \leftarrow \mathsf{MAC}(\mathsf{k}_i, y)$. Forward $(y, \sigma_1, \ldots, \sigma_n)$ as the response to Sim_Φ, which will return $\{\phi_2^h\}_{h \in [m] \setminus \{C \cup C'\}}$. For each $h \in [m] \setminus \{C' \cup C\}$, reply to the simulator Sim_{Π_h} with ϕ_2^h, thus obtaining $\{\pi_{h,3}^i, \tilde{\pi}_{h,2}^i, \pi_{h,3}^{\mathsf{prg},i}, \tilde{\pi}_{h,2}^{\mathsf{prg},i}\}_{i \in H}$
9. For each $i \in H, h \in [m]$ compute $\tilde{\mathsf{C}}_h^i, (\mathsf{lab}^{i,h,c})_{c \in [n] \setminus \{i\}} \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, (\tilde{\pi}_2^i, \tilde{\pi}_2^{\mathsf{prg},i}))$
10. Send $(\mathsf{labels}, \{\mathsf{lab}^{i,h,c}\}_{i \in H, h \in [m], c \in [n] \setminus \{i\}})$ to the right interface. Upon receiving $\{\mathsf{ot}_4^{j,i}\}_{j \in M, i \in H}$ from the right interface, for each $i \in H$ send $\{\pi_{h,3}^i, \pi_{h,3}^{\mathsf{prg},i}, \tilde{\mathsf{C}}_h^i\}_{h \in [m]}, \{\mathsf{ot}_4^{j,i}\}_{j \in [n] \setminus \{i\}}, (\mathsf{otinp}^i, \rho^i)$ to $\mathcal{A}$

Output

1. If some party aborted at the end of the third round, then instruct the ideal functionality to send $\perp$ to all honest parties.
2. For each $h \in [m]$, compute ϕ_2^h using the output out_{Π_h}
3. Run out_Φ on $(\phi_2^1, \ldots, \phi_2^m)$ to compute $(y', \sigma_1, \ldots, \sigma_n)$.
4. For each $i \in H$, if $(y, \sigma_i) \neq (y', \sigma_i')$ then Sim instructs the trusted functionality to send $\perp$ to P_i. Else, it instructs the functionality to deliver the output to P_i.
5. Return what $\mathcal{A}$ returns.

Running time of the Simulator. The running time of this simulator is dominated by the simulators $\mathsf{Sim}_{\mathsf{wI}}$ and $\mathsf{Sim}_{\mathsf{ot}}$, each running in expected polynomial time. The simulators for the inner-protocol Sim_{Π_h} and the simulator for the outer protocol Φ all run in polynomial time. Given that $\mathsf{Sim}_{\mathsf{wI}}$ and $\mathsf{Sim}_{\mathsf{ot}}$ are executed one after the other, we obtain that the total running time of the simulator is still expected polynomial time. The proof is divided into two main parts as we distinguish the case where the adversary makes the simulator abort in the third round and the case where the simulator does not abort. We start by proving security in the first case.

Security against aborting adversaries. Consider the case where the adversary provides a set of messages in the first three rounds that make the honest parties abort. We note that the honest parties do not use their inputs to compute the first and the second rounds. Hence, if the adversary aborts in the first or the second round, the inputs of the honest parties are trivially protected. The only interesting case occurs when the adversary provides an aborting transcript in the third round, after having received the third round message of the honest parties. However, we note that in the event that all the honest parties abort, then these honest parties will just send an abort message in the fourth round. In particular, the honest parties will not disclose the input-randomness pair used to generate the messages of the nRmS OT $\mathsf{OT}_{\vec{S},\vec{R}}$ protocol. Hence, the input of the parties remains protected, as the only protocol message that encodes the input of an honest party is $\tilde{\pi}^i_{h,1}$ (with $i \in H$ and $h \in [m]$), which is used only as an input to the i-th receiver of the nRmS OT protocol. Hence, we can rely on the security of $\mathsf{OT}_{\vec{S},\vec{R}}$, to argue that the view of the adversary in the real game is indistinguishable from the view of an adversary in which the honest receivers, running $\mathsf{OT}_{\vec{S},\vec{R}}$, use a default input instead of $\tilde{\pi}^i_{h,1}$, for each $i \in H$ and $h \in [m]$. We finally observe that this reduction is immediate as there are no other rewinds that could affect it. Indeed, $\mathsf{Sim}^1_{\mathsf{wl}}$ does not perform any rewind in the case when the adversary aborts in the third round.

Security against sometimes aborting adversaries. Assuming that the adversary provides an accepting third round for at least one honest party, we want to argue that our protocol is secure. The proof proceeds by a sequence of hybrid experiments that we summarize below.

Hybrid H_0: This corresponds to the real world experiment (we refer to Definition 2.6 for the formal definition of real-ideal world) where the adversary $\mathcal{A}$ corrupts the parties with indices M and interacts with the honest parties indexed by H.

Hybrid H_1: This hybrid is the same as the previous one with the exception that the messages of the watchlist protocol are simulated. We refer to Figure 5.5 for the formal description of the hybrid. The indistinguishability between the two hybrids follows from the security of WL. Note that in this part of the proof we are considering adversaries that provide an accepting third round, and that WL is secure against this type of adversaries.

Hybrid H_2: This hybrid is the same as the previous one with the exception that the messages of $\mathsf{OT}_{\vec{S},\vec{R}}$, where the honest parties act as senders, are simulated (we refer to Figure 5.6 for the formal description of this hybrid). The reduction to the security of $\mathsf{OT}_{\vec{S},\vec{R}}$ is almost straightforward, but we need to argue that the rewinds performed by the simulator of WL do not perturb the reduction. This can be easily argued since we consider all the receivers in the reduction to be corrupted, and only a subset of the senders (those controlled by the honest parties) as being honest. Note that the challenger for the security game defined in Definition 3.6 sends messages only in the second (in the third we implicitly send the public keys needed to realize the private channel as discussed in Section 3) and the last round. Hence, in the rewinding threads generated by the simulator of WL, the reduction can generate the second and the third rounds using fresh randomness. The messages of the challenger will be used only in the output thread generated by WL. We finally note that in this part of the proof we do not care about protecting the inputs of the honest receivers since, in the case where the adversary provides an accepting transcript with non-negligible probability, the input and the randomness of the receivers will be anyway disclosed in the last round. Therefore, an adversary that does not abort in the third round has always access to the message $\tilde{\pi}$ used by the honest parties as the receiver's input of $\mathsf{OT}_{\vec{S},\vec{R}}$.

Hybrid H_3: This hybrid is the same as the previous one with the exception of the following: initialize an empty set C'. For each $h \in [m]$, check if there exists some $j \in H$ such that for every $i \in M, y_{i,j}$ contains the randomness that explains the messages sent by the corrupted parties in Π_h as well as the correct PRG computations. If not, add h to C'. If $|C'| > \lambda n^2$ stop and instruct all the honest parties to output $\perp$, else continue. In Lemma 5.2, we argue that this and the previous hybrid are statistically indistinguishable.

Hybrid H_4: This hybrid is the same as the previous one with the exception that the garbled circuits generated by the honest parties are simulated. This results in a different interaction of the augmented machine with the simulators of wl and ot. We provide a formal description of the augmented machine in Figure 5.8. At a high level, the main difference between this and the previous hybrid is the following: let $\mathsf{otinp}^i = (\tilde{\pi}^i_{h,1})_{h\in[m]}$ be the input extracted from $\mathsf{Sim}_{\mathsf{ot}}$, for each $i \in M$. The hybrid computes the third round message of $\mathit{\Pi}$ by running Π_2^{inp} (this can be done because the messages $\{\tilde{\pi}^i_{h,1}\}_{h\in[m]}$ are now available in the clear since $\mathsf{Sim}_{\mathsf{ot}}$ extracted them). Let $(\tilde{\pi}^i_2, \tilde{\pi}_2^{\mathsf{prg},i})$ denote the output obtained from Π_2^{inp} for each $i \in H$, then the h-th garbled circuit for the i-th party is computed using the simulator of the garbled circuit on the input $(\tilde{\pi}^i_2, \tilde{\pi}_2^{\mathsf{prg},i})$. The indistinguishability between the previous and this hybrid comes from the security of the garbled circuit scheme.

Hybrid H_5: This hybrid is the same as the previous one with the the following differences. After the extraction phase, the hybrid, with respect to the messages of the inner-protocol executions (Π), behaves exactly like described in Figure 5.4. During the extraction phase, the messages of Π are simulated. After the extraction, we construct the sets C' and C in the same way as in the simulator of Figure 5.4, and then compute the second and third round messages of the inner protocol executions, the hybrid acts as the honest parties would for all the inner protocols with indices in C. All the other inner protocol messages are computed using the simulator $\mathsf{Sim}_{\mathit{\Pi}_h}$ for each $h \notin C$. To compute the inner protocol messages for the fourth round, the hybrid acts as follows:

- For each $h \in C$ compute the messages of Π_h as the honest parties would.
- For each $h \in C'$ feed the simulator with the inputs that the honest parties would have used to compute the messages of the h-th inner protocol.
- For each $h \in [n] \setminus \{C \cup C'\}$ feed the simulator $\mathsf{Sim}_{\mathit{\Pi}_h}$ with the randomness provided by the adversary in $y_{i,j}$ (for each $i \in H$, $j \in M$). When $\mathsf{Sim}_{\mathit{\Pi}_h}$ makes a query to its ideal functionality, reply to the query by evaluating Φ_2^h using the inputs provided by the simulator $\mathsf{Sim}_{\mathit{\Pi}_h}$ and the input of the honest parties.

The indistinguishability between this and the previous hybrid comes from the security of the inner protocol.

Hybrid H_6: This hybrid is the same as the previous one with the exception that the messages of the outer protocol are not computed using Φ_2^h, but are simulated exactly as described in Sim (Figure 5.3). The indistinguishability between this and the previous hybrid follows from the security against adaptive corruption of Φ.

Hybrid H_7: In this hybrid we make the following change: in the output phase, we recover $(y', \sigma_1, \ldots, \sigma_1)$ as in the previous hybrid and then check if $y' = y$ and if for each $i \in H$, $\sigma'_i = \sigma_i$. For every $i \in H$, for which the above check passes, the ideal functionality is instructed to deliver the output of P_i. For all the remaining parties, it instructs them to abort. In Lemma 5.3 we show that H_7 and H_6 are statistically indistinguishable by relying on the security of the MAC scheme. The formal description of the hybrid corresponds to the description of the simulator. The proof concludes with the observation that the output of H_7 is identically distributed to the output of the ideal world execution.

Lemma 5.2 (Indistinguishability of hybrids H_1 and H_2). *The hybrids H_1 and H_2 are statistically indistinguishable.*

Proof. This proof works exactly in the same way as the proof of [IKSS21, Claim 6.7]. We recap it here verbatim for completeness. Fix any honest party P_i. Note that P_i aborts in H_2 if $|K_i \cap C'| \neq 0$. We show that if $|C'| > \lambda n^2$ then the probability of $|K_i \cap C'| = 0$ is negligible.

Note that K_i is distributed as a random subset of $[m]$ of size λ. We now upper bound the probability that $|K_i \cap C'| = 0$.

$$\begin{aligned}
\Pr[|K_i \cap C'| = 0] &= \frac{\binom{m-|C'|}{\lambda}}{\binom{m}{\lambda}} \\
&< \frac{\binom{m-\lambda n^2}{\lambda}}{\binom{m}{\lambda}} \\
&= \frac{(m-\lambda n^2)!}{m!} \frac{(m-\lambda)!}{(m-\lambda-\lambda n^2)!} \\
&< (1-\lambda/m)^{\lambda n^2} \\
&< 2^{-O(\lambda)}
\end{aligned}$$

The lemma now follows from a standard union bound over the set of all honest parties. □

Lemma 5.3 (Indistinguishability of hybrids H_6 and H_7). *Let MAC be a secure MAC scheme, then the hybrids H_6 and H_7 are indistinguishable.*

Proof. The only difference between the two hybrids is that in H_7 it is checked that $y = y'$ and that for each $i \in H$, $\sigma'_i = \sigma_i$. For the honest parties where this check fails, they are instructed to abort. All other honest parties are instructed to output y. In hybrid H_6, on the other hand, the honest parties are instructed to do the MAC verification and, depending on the result, they abort or output y'. If an honest party does not abort in H_7, then the correctness of the verification procedure implies that it does not abort in H_6. Assume that there exists an honest party P_i that aborts in H_7, but that does not abort with non-negligible probability in H_6, then this means that $(y', \sigma'_i) \neq (y, \sigma_i)$ and that the verification procedure on (y', σ'_i) outputs 1. This contradicts the security of the MAC scheme and therefore the theorem follows. □

Figure 5.5: Hybrid H_1

1. Run $\mathsf{Sim}^1_{\mathsf{WL}}$ against the adversary $\mathcal{M}_{\mathsf{H}_{1,2}}$ (defined in Figure 5.7) by sending messages on its right interface with the following modifications. When $\mathcal{M}_{\mathsf{H}_{1,2}}$ waits for the messages of $\mathsf{OT}_{\vec{S},\vec{R}}$, computed on behalf of the honest senders in the second round, compute these messages as the honest senders of $\mathsf{OT}_{\vec{S},\vec{R}}$ do, thus obtaining $\{\mathsf{ot}_2^{i,j}\}_{i\in H, j\in M}$ (note that no input is needed to the sender to compute the second round of $\mathsf{OT}_{\vec{S},\vec{R}}$) and send them to the right interface of $\mathsf{OT}_{\vec{S},\vec{R}}$ along with the messages of $\mathsf{Sim}^1_{\mathsf{WL}}$.
2. If $\mathsf{Sim}^1_{\mathsf{WL}}$ stops and returns $\perp$, then instruct $\mathcal{M}_{\mathsf{H}_{1,2}}$ to abort, and return what $\mathcal{M}_{\mathsf{H}_{1,2}}$ returns. Else, $\mathsf{Sim}^1_{\mathsf{WL}}$ returns the output $(\{x_{i,j}\}_{i\in M, j\in H}, 1^k)$, and the behavior of $\mathcal{F}^{\mathsf{list}}_{OT^{\lambda,m}}$ on input $\{x_{i,j}\}_{i\in M, j\in H}$ is emulated to obtain $\{\mathsf{out}^{\kappa}_{i,j}\}_{i\in H, j\in M, \kappa\in[k]}$.
3. Run $\mathsf{Sim}^2_{\mathsf{WL}}(\{\mathsf{out}^{\kappa}_{i,j}\}_{i\in M, j\in H, \kappa\in[k]}, z)$ against the adversary $\mathcal{M}_{\mathsf{H}_{1,2}}$.
4. Let $(\iota, \{y_{i,j}\}_{i\in[M], j\in H}, \mathrm{View})$ be the output of $\mathsf{Sim}^2_{\mathsf{WL}}$, send $\{y_{i,j}\}_{i\in[M], j\in H}$ to the right interface of $\mathcal{M}_{\mathsf{H}_{1,2}}$ along with the third round of $\mathsf{Sim}^2_{\mathsf{WL}}$.
5. When $\mathcal{M}_{\mathsf{H}_{1,2}}$ outputs $(\mathsf{labels}, \{\mathsf{lab}_i\}_{i\in H})$ on its right interface, then, for each i:
   (a) Parse lab_i as $\{\mathsf{lab}_0^{j,k,h}, \mathsf{lab}_1^{j,k,h}\}_{j\in[n]\setminus\{i\}, k\in[\ell], h\in[m]}$
   (b) For each $j \in [n] \setminus \{i\}$, compute $\mathsf{ot}_4^{j,i} \leftarrow \mathsf{OT}^j_{S,4}(1^\lambda, ((\mathsf{lab}_0^{j,k,h}, \mathsf{lab}_1^{j,k,h})_{k\in[\ell]})_{h\in[m]}, \mathsf{ot}(2))$.
6. Send $\{\mathsf{ot}_2^{i,j}\}_{i\in H, j\in M}$ to $\mathcal{M}_{\mathsf{H}_{1,2}}$.
7. Return what $\mathcal{M}_{\mathsf{H}_{1,2}}$ returns.

Figure 5.6: Hybrid H_2

1. Run $\mathsf{Sim}^1_{\mathsf{WL}}$ against the adversary $\mathcal{M}_{\mathsf{H}_{1,2}}$ (defined in Figure 5.7) by sending messages on its right interface with the following modifications. When $\mathcal{M}_{\mathsf{H}_{1,2}}$ waits for the messages of $\mathsf{OT}_{\vec{S},\vec{R}}$, computed on behalf of the honest senders in the second round, compute these messages as the honest senders of $\mathsf{OT}_{\vec{S},\vec{R}}$ do, thus obtaining $\{\mathsf{ot}_2^{i,j}\}_{i\in H, j\in M}$ (note that no input is needed to the sender to compute the second round of $\mathsf{OT}_{\vec{S},\vec{R}}$) and send them to the right interface of $\mathsf{OT}_{\vec{S},\vec{R}}$ along with the messages of $\mathsf{Sim}^1_{\mathsf{WL}}$.
2. If $\mathsf{Sim}^1_{\mathsf{WL}}$ stops and returns $\perp$, then instruct $\mathcal{M}_{\mathsf{H}_{1,2}}$ to abort, and return what $\mathcal{M}_{\mathsf{H}_{1,2}}$ returns. Else, $\mathsf{Sim}^1_{\mathsf{WL}}$ returns the output $(\{x_{i,j}\}_{i\in M, j\in H}, 1^k)$, and the behavior of $\mathcal{F}^{\mathsf{list}}_{OT^{\lambda,m}}$ is emulated on input $\{x_{i,j}\}_{i\in M, j\in H}$ to obtain $\{\mathsf{out}^{\kappa}_{i,j}\}_{i\in H, j\in M, \kappa\in[k]}$.
3. Run $\mathsf{Sim}_{\mathsf{ot}}$ against the adversary $\mathcal{M}_{\mathsf{H}_{1,2}}$, and any time that $\mathcal{M}_{\mathsf{H}_{1,2}}$ requires a message of the watchlist protocol compute it accordingly to $\mathsf{Sim}^1_{\mathsf{WL}}$. If $\mathsf{Sim}_{\mathsf{ot}}$ returns $\perp$, instruct the ideal functionality to abort. Else, if $\mathsf{Sim}_{\mathsf{ot}}$ returns $\{\mathsf{otinp}^i\}_{i\in M}$, continue as follows.
4. Run $\mathsf{Sim}^2_{\mathsf{WL}}(\{\mathsf{out}^{\kappa}_{i,j}\}_{i\in M, j\in H, \kappa\in[k]}, z)$ against the adversary $\mathcal{M}_{\mathsf{H}_{1,2}}$.
5. Let $(\iota, \{y_{i,j}\}_{i\in[M], j\in H}, \mathrm{View})$ be the output of $\mathsf{Sim}^2_{\mathsf{WL}}$, send $\{y_{i,j}\}_{i\in[M], j\in H}$ to the right interface of $\mathcal{M}_{\mathsf{H}_{1,2}}$ along with the third round of $\mathsf{Sim}^2_{\mathsf{WL}}$ to $\mathcal{M}_{\mathsf{H}_{1,2}}$.

6. When $\mathcal{M}_{\mathsf{H}_{1,2}}$ outputs $(\mathsf{labels}, \{\mathsf{lab}_i\}_{i \in H})$ on its right interface do the following
   (a) For each $i \in H$ parse lab_i as $\{\mathsf{lab}_0^{j,k,h}, \mathsf{lab}_1^{j,k,h}\}_{j \in [n] \setminus \{i\}, k \in [\ell], h \in [m]}$
      - For each $j \in M$ parse otinp_j as $\{\mathsf{otinp}_{j,c}\}_{c \in H}$ and set $\mathsf{lab}_{i,j} := \{\mathsf{lab}^{j,k,h}_{\mathsf{otinp}_{j,i}}\}_{k \in [\ell], h \in [m]}$.
7. Reply to $\mathsf{Sim}_{\mathsf{ot}}$, acting on behalf of its ideal functionality, with the values $\{\mathsf{lab}_{i,j}\}_{i \in H, j \in M}$, and, upon receiving $\{\mathsf{ot}_2^{i,j}\}_{i \in H, j \in M}$, forward it to $\mathcal{M}_{\mathsf{H}_{1,2}}$.
8. Return what $\mathcal{M}_{\mathsf{H}_{1,2}}$ returns.

Figure 5.7: $\mathcal{M}_{\mathsf{H}_{1,2}}$

This machine has oracle access to $\mathcal{A}$, the adversary attacking our MPC protocol.

Round 1:

1. Upon receiving $\{\mathsf{wl}_1^i\}_{i \in H}$ from the right interface, for each $i \in H$ sample $\rho^i \leftarrow \{0,1\}^\lambda$, compute $\mathsf{ot}_1^i \leftarrow \mathsf{OT}^i_{R,1}(1^\lambda; \rho^i)$ and send $(\mathsf{wl}_1^i, \mathsf{ot}_1^i)$ to $\mathcal{A}$.
2. Upon receiving $(\mathsf{wl}_1^i, \mathsf{ot}_1^i)_{i \in M}$ from $\mathcal{A}$ forward these messages on the right interface.

Round 2. Upon receiving $\{(\mathsf{wl}_2^i, y_{i,j}), \mathsf{ot}_2^{i,j}\}_{i \in H, j \in M}$ from the right interface, for each $i \in H, j \in M$ parse $y_{i,j}$ as $\{r_{i,h}, r_{i,h}^{\mathsf{prg}}, s_{i,h}, S_{i,h}\}_{h \in [m]}$ and do the following:

1. For each $h \in [m]$ compute $\pi^i_{h,1} := \Pi^{\mathsf{no\text{-}inp}}_{h,1}(1^\lambda, i; r_{i,h})$, $\pi^{\mathsf{prg},i}_{h,1} := \Pi^{\mathsf{no\text{-}inp}}_{h,1}(1^\lambda, i; r^{\mathsf{prg}}_{i,h})$ for all $h \in [m]$.
2. For each $h \in [m]$ send $\mathsf{wl}_2^i, \{\pi^i_{h,1}, \pi^{\mathsf{prg},i}_{h,1}\}_{h \in [m]}, \{\mathsf{ot}_2^{i,j}\}_{j \in [n] \setminus \{i\}}$ to $\mathcal{A}$.

Round 3. Upon receiving $\{\mathsf{wl}_2^i, \{\pi^i_{h,1}, \pi^{\mathsf{prg},i}_{h,1}\}_{h \in [m]}, \{\mathsf{ot}_2^{i,j}\}_{j \in H}\}_{i \in H}$ from $\mathcal{A}$ do the following:

1. Forward $\{\mathsf{wl}_2^i\}_{i \in M}$ to the right interface. Upon receiving $\{\mathsf{wl}_3^i\}_{i \in H}$ from the right interface, for each $i \in H$ do the following.
2. Sample $\mathsf{k}_i \leftarrow \{0,1\}^*$, set $z_i := (x_i, \mathsf{k}_i)$, compute $(\phi_1^{i \to 1}, \ldots, \phi_1^{i \to m}) \leftarrow \Phi_1(1^\lambda, i, z_i)$.
3. For each $h \in [m]$ do the following
   (a) $(z_h, \pi^i_{h,2}) \leftarrow \Pi_2^{\mathsf{no\text{-}inp}}(i, 1^\lambda, \pi_h(1); r_{i,h})$, $\tilde{\pi}^i_{h,1} \leftarrow \Pi_1^{\mathsf{inp}}(i, \phi_1^{i \to 1}, z_h)$
   (b) $(z_h^{\mathsf{prg}}, \pi^{i,\mathsf{prg}}_{h,2}) \leftarrow \Pi_2^{\mathsf{no\text{-}inp}}(i, 1^\lambda, \pi_h^{\mathsf{prg}}(1); r^{\mathsf{prg}}_{i,h})$, $\tilde{\pi}^{i,\mathsf{prg}}_{h,1} \leftarrow \Pi_1^{\mathsf{inp}}(i, (s_{i,h}, S_{i,h}), z_h^{\mathsf{prg}})$
4. Set $\mathsf{otinp}^i := (\tilde{\pi}^i_{h,1})_{h \in [m]}$, compute $\mathsf{ot}_3^i \leftarrow \mathsf{OT}^i_{R,3}(\mathsf{otinp}^i, \mathsf{ot}^i(2); \rho^i)$.
5. Send $\mathsf{wl}_3^i, \{\pi^i_{h,2}, \pi^{\mathsf{prg},i}_{h,2}, \tilde{\pi}^{i,\mathsf{prg}}_{h,1}\}_{h \in [m]}, \mathsf{ot}_3^i$ to $\mathcal{A}$.

Round 4. Upon receiving $\{\mathsf{wl}_3^i, \{\pi^i_{h,2}, \pi^{\mathsf{prg},i}_{h,2}, \tilde{\pi}^{i,\mathsf{prg}}_{h,1}\}_{h \in [m]}, \mathsf{ot}_3^i\}_{i \in M}$ from $\mathcal{A}$, forward $\{\mathsf{wl}_3^i, \mathsf{ot}_3^i\}_{i \in M}$ to the right interface. On receiving $(\{y_{i,j}\}_{i \in M, j \in H})$ from the right interface (where $\{y_{i,j}\}_{i \in M, j \in H}$ represents the extracted values related to the watchlist senders' inputs (recall that $y_{i,j} = \{r_{i,h}, r^{\mathsf{prg}}_{i,h}, s_{i,h}, S_{i,h}\}_{h \in [m]}$) continue as follows.

1. For all $j \in M$ and $h \in K_i$, check that:
   - The PRG computations in $(s_{j,h}, S_{j,h})$ are correct.
   - Compute $\pi^{\star j}_{h,1} := \Pi^{\mathsf{no\text{-}inp}}_{h,1}(1^\lambda, j, \pi_h(0); r_{j,h})$ and $\pi^{\star j,\mathsf{prg}}_{h,1} := \Pi^{\mathsf{no\text{-}inp}}_{h,1}(1^\lambda, j, \pi_h(0); r^{\mathsf{prg}}_{j,h})$, check that $\pi^{\star j}_{h,1} = \pi^j_{h,1}$ and $\pi^{\star j,\mathsf{prg}}_{h,1} = \pi^{j,\mathsf{prg}}_{h,1}$.
   - Compute $(z_h, \pi^{j\star}_{h,2}) := \Pi_2^{\mathsf{no\text{-}inp}}(j, 1^\lambda, \pi_h(1); r_{j,h})$ check that $\pi^{j\star}_{h,2} = \pi^j_{h,2}$.

   - Compute $(z_h^{\mathsf{prg}}, \pi_{h,2}^{\mathsf{prg},j\star}) := \Pi_2^{\mathsf{no\text{-}inp}}(j, 1^\lambda, \pi_h(1), r_{j,h}^{\mathsf{prg}})$ and $\tilde{\pi}_{1,h}^{\mathsf{prg},j\star} := \Pi_1^{\mathsf{inp}}(j, (s_{j,h}, S_{j,h}), z_h^{\mathsf{prg}})$, check that $\pi_{h,2}^{\mathsf{prg},j\star} = \pi_{h,2}^{\mathsf{prg},j}$ and $\tilde{\pi}_{1,h}^{\mathsf{prg},j\star} = \tilde{\pi}_{1,h}^{\mathsf{prg},j}$

2. If any of the above checks fail, output $\perp$, else, for each $i \in H$, do the following
   (a) For each $h \in [m]$ compute $\tilde{\mathsf{C}}_h^i, \{\mathsf{lab}_0^{j,k,h}, \mathsf{lab}_1^{j,k,h}\}_{j\in[n]\setminus\{i\},k\in[\ell],h\in[m]} \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{C}_h[\phi_1^{i\to h}, s_{i,h}, S_{i,h}, z_h, z_h^{\mathsf{prg}}, \{\tilde{\pi}_{h,1}^{\mathsf{prg},j}\}_{j\in[n]}])$
   (b) Set $\mathsf{lab}_i := \{\mathsf{lab}_0^{j,k,h}, \mathsf{lab}_1^{j,k,h}\}_{j\in[n]\setminus\{i\},k\in[\ell],h\in[m]}$
   (c) For all $h \in [m]$, compute $\pi_{h,3}^i := \Pi_{h,3}^{\mathsf{no\text{-}inp}}(1^\lambda, i, \pi_h(2); r_{i,h})$ $\pi_{h,3}^{\mathsf{prg},i} := \Pi_{h,3}^{\mathsf{no\text{-}inp}}(1^\lambda, i, \pi_h(2); r_{i,h}^{\mathsf{prg}})$.
3. Send $(\mathsf{labels}, \{\mathsf{lab}_i\}_{i\in H})$ to the right interface. Upon receiving $\{\mathsf{ot}_4^{j,i}\}_{j\in M, i\in H}$ from the right interface, for each $i \in H$, send $\{\pi_{h,3}^i, \pi_{h,3}^{\mathsf{prg},i}, \tilde{\mathsf{C}}_h^i\}_{h\in[m]}, \{\mathsf{ot}_4^{j,i}\}_{j\in[n]\setminus\{i\}}, (\mathsf{otinp}^i, \rho^i)$ to $\mathcal{A}$

Output

1. For each $h \in [m]$, compute ϕ_2^h using the output out_{Π_h}.
2. Run out_Φ on $(\phi_2^1, \dots, \phi_2^m)$ to compute $(y', \sigma_1, \dots, \sigma_n)$.
3. Return y' and what $\mathcal{A}$ returns.

Figure 5.8: $\mathcal{M}_{\mathsf{H}_4}$

This machine has oracle access to $\mathcal{A}$, the adversary attacking our MPC protocol.

Round 1:

1. Upon receiving $\{\mathsf{wl}_1^i\}_{i\in H}$ from the right interface, for each $i \in H$ sample $\rho^i \leftarrow \{0,1\}^\lambda$, compute $\mathsf{ot}_1^i \leftarrow \mathsf{OT}_{R,1}^i(1^\lambda; \rho^i)$ and send $(\mathsf{wl}_1^i, \mathsf{ot}_1^i)$ to $\mathcal{A}$.
2. Upon receiving $(\mathsf{wl}_1^i, \mathsf{ot}_1^i)_{i\in M}$ from $\mathcal{A}$ forward these messages to the right interface.

Round 2. Upon receiving $\{(\mathsf{wl}_2^i, y_{i,j}), \mathsf{ot}_2^{i,j}\}_{i\in H, j\in M}$ from the right interface, for each $i \in H, j \in M$, parse $y_{i,j}$ as $\{r_{i,h}, r_{i,h}^{\mathsf{prg}}, s_{i,h}, S_{i,h}\}_{h\in[m]}$ and do the following:

1. For each $h \in [m]$, compute $\pi_{h,1}^i := \Pi_{h,1}^{\mathsf{no\text{-}inp}}(1^\lambda, i; r_{i,h})$, $\pi_{h,1}^{\mathsf{prg},i} := \Pi_{h,1}^{\mathsf{no\text{-}inp}}(1^\lambda, i; r_{i,h}^{\mathsf{prg}})$ for all $h \in [m]$.
2. For each $h \in [m]$, send $\mathsf{wl}_2^i, \{\pi_{h,1}^i, \pi_{h,1}^{\mathsf{prg},i}\}_{h\in[m]}, \{\mathsf{ot}_2^{i,j}\}_{j\in[n]\setminus\{i\}}$ to $\mathcal{A}$.

Round 3. Upon receiving $\{\mathsf{wl}_2^i, \{\pi_{h,1}^i, \pi_{h,1}^{\mathsf{prg},i}\}_{h\in[m]}, \{\mathsf{ot}_2^{i,j}\}_{j\in H}\}_{i\in H}$ from $\mathcal{A}$ do the following:

1. Forward $\{\mathsf{wl}_2^i\}_{i\in M}$ to the right interface. Upon receiving $\{\mathsf{wl}_3^i\}_{i\in H}$ from the right interface, for each $i \in H$ do as follows.
2. Sample $\mathsf{k}_i \leftarrow \{0,1\}^*$, set $z_i := (x_i, \mathsf{k}_i)$, compute $(\phi_1^{i\to 1}, \dots, \phi_1^{i\to m}) \leftarrow \Phi_1(1^\lambda, i, z_i)$.
3. For each $h \in [m]$ do the following:
   (a) $(z_h, \pi_{h,2}^i) \leftarrow \Pi_2^{\mathsf{no\text{-}inp}}(i, 1^\lambda, \pi_h(1); r_{i,h})$, $\tilde{\pi}_{h,1}^i \leftarrow \Pi_1^{\mathsf{inp}}(i, \phi_1^{i\to 1}, z_h)$
   (b) $(z_h^{\mathsf{prg}}, \pi_{h,2}^{i,\mathsf{prg}}) \leftarrow \Pi_2^{\mathsf{no\text{-}inp}}(i, 1^\lambda, \pi_h^{\mathsf{prg}}(1); r_{i,h}^{\mathsf{prg}})$, $\tilde{\pi}_{h,1}^{i,\mathsf{prg}} \leftarrow \Pi_1^{\mathsf{inp}}(i, (s_{i,h}, S_{i,h}), z_h^{\mathsf{prg}})$
4. Set $\mathsf{otinp}^i := (\tilde{\pi}_{h,1}^i)_{h\in[m]}$, compute $\mathsf{ot}_3^i \leftarrow \mathsf{OT}_{R,3}^i(\mathsf{otinp}^i, \mathsf{ot}^i(2); \rho^i)$.
5. Send $\mathsf{wl}_3^i, \{\pi_{h,2}^i, \pi_{h,2}^{\mathsf{prg},i}, \tilde{\pi}_{h,1}^{i,\mathsf{prg}}\}_{h\in[m]}, \mathsf{ot}_3^i$ to $\mathcal{A}$.

Round 4. Upon receiving $\{\mathsf{wl}_3^i, \{\pi_{h,2}^i, \pi_{h,2}^{\mathsf{prg},i}, \tilde{\pi}_{h,1}^{i,\mathsf{prg}}\}_{h\in[m]}, \mathsf{ot}_3^i\}_{i\in M}$ from $\mathcal{A}$, forward $\{\mathsf{wl}_3^i, \mathsf{ot}_3^i\}_{i\in M}$ to the right interface. On receiving $(\{y_{i,j}\}_{i\in M, j\in H}, \{\mathsf{ot}_4^{i,j}\}_{i\in H, j\in[n]\setminus\{i\}}, \{\mathsf{otinp}^i\}_{i\in M})$ from the right interface (where $\{y_{i,j}\}_{i\in M,j\in H}$ represents the extracted values related to the watchlist senders' inputs (recall that $y_{i,j} = \{r_{i,h}, r_{i,h}^{\mathsf{prg}}, s_{i,h}, S_{i,h}\}_{h\in[m]}$), and $\{\mathsf{otinp}^i\}_{i\in M}$ represent the receivers' input used by the adversary in the execution of nRmS), parse otinp^i as $(\tilde{\pi}_{h,1}^i)_{h\in[m]}$ (with $i \in M$) and continue as follows.

1. Initialize the empty set C'. For each $h \in [m]$, check if there exists some $j \in H$ such that for every $i \in M$, $y_{i,j}$ contains the randomness that explains the messages sent by corrupted parties in Π_h as well as the correct PRG computations. If not, it adds h to C'. If $|C'| > \lambda n^2$ stop and and instructs all the honest parties to output $\perp$, else continue.
2. For all $j \in [n] \setminus \{i\}$ and $h \in K_i$, check that:
   - The PRG computations in $(s_{i,h}, S_{i,h})$ are correct.
   - Compute $\pi_{h,1}^{\star j} := \Pi_{h,1}^{\mathsf{no\text{-}inp}}(1^\lambda, j, \pi_h(0); r_{j,h})$ and $\pi_{h,1}^{\star j,\mathsf{prg}} := \Pi_{h,1}^{\mathsf{no\text{-}inp}}(1^\lambda, j, \pi_h(0); r_{j,h}^{\mathsf{prg}})$, check that $\pi_{h,1}^{\star j} = \pi_{h,1}^{j}$ and $\pi_{h,1}^{\star j,\mathsf{prg}} = \pi_{h,1}^{j,\mathsf{prg}}$.
   - Compute $(z_h, \pi_{h,2}^{j\star}) := \Pi_2^{\mathsf{no\text{-}inp}}(j, 1^\lambda, \pi_h(1); r_{j,h})$ check that $\pi_{h,2}^{j\star} = \pi_{h,2}^{j}$.
   - Compute $(z_h^{\mathsf{prg}}, \pi_{h,2}^{\mathsf{prg},j\star}) := \Pi_2^{\mathsf{no\text{-}inp}}(j, 1^\lambda, \pi_h(1), r_{j,h}^{\mathsf{prg}})$ and $\tilde{\pi}_{1,h}^{\mathsf{prg},j\star} := \Pi_1^{\mathsf{inp}}(j, (s_{j,h}, S_{j,h}), z_h^{\mathsf{prg}})$, check that $\pi_{h,2}^{\mathsf{prg},j\star} = \pi_{h,2}^{\mathsf{prg},j}$ and $\tilde{\pi}_{1,h}^{\mathsf{prg},j\star} = \tilde{\pi}_{1,h}^{\mathsf{prg},j}$
3. If any of the above checks fail, output $\perp$.
4. For each $h \in C, i \in H$
   (a) Set $\tilde{\pi} = \{\tilde{\pi}_{h,1}^{j}, \tilde{\pi}_{h,1}^{\mathsf{prg}}, \}_{j\in[n]\setminus\{i\}}$
   (b) $\pi_{h,3}^i := \Pi_{h,3}^{\mathsf{no\text{-}inp}}(1^\lambda, i, \pi_h(2); r_{i,h})$
   (c) $\pi_{h,3}^{\mathsf{prg},i} := \Pi_{h,3}^{\mathsf{no\text{-}inp}}(1^\lambda, i, \pi_h(2); r_{i,h}^{\mathsf{prg}})$
   (d) $\tilde{\pi}_{h,2}^i \leftarrow \Pi_2^{\mathsf{inp}}(i, \phi_1^{i\to h}, z_{i,h}, \tilde{\pi})$,
   (e) $\tilde{\pi}_2^{\mathsf{prg},i} \leftarrow \Pi_2^{\mathsf{inp}}(i, (s_{i,h}, S_{i,h}), z_{i,h}^{\mathsf{prg}}, \tilde{\pi})$.
5. For each $i \in H, h \in [m]$ compute $\tilde{\mathsf{C}}_h^i, (\mathsf{lab}^{i,h,c})_{c\in[n]\setminus\{i\}} \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, (\tilde{\pi}_2^i, \tilde{\pi}_2^{\mathsf{prg},i}))$
6. Send $(\mathsf{labels}, \{\mathsf{lab}^{i,h,c}\}_{i\in H, h\in[m], c\in[n]\setminus\{i\}})$ to the right interface. Upon receiving $\{\mathsf{ot}_4^{j,i}\}_{j\in M, i\in H}$ from the right interface, for each $i \in H$ send $\{\pi_{h,3}^i, \pi_{h,3}^{\mathsf{prg},i}, \tilde{\mathsf{C}}_h^i\}_{h\in[m]}, \{\mathsf{ot}_4^{j,i}\}_{j\in[n]\setminus\{i\}}, (\mathsf{otinp}^i, \rho^i)$ to $\mathcal{A}$.

Output

1. For each $h \in [m]$, compute ϕ_2^h using the output out_{Π_h}
2. Run out_Φ on $(\phi_2^1, \dots, \phi_2^m)$ to compute $(y', \sigma_1, \dots, \sigma_n)$.
3. Return y' and what $\mathcal{A}$ returns.

Corollary 5.4. *The protocol of Figure 5.1 makes black-box use of two-round OTs with statistical sender security.*

Before arguing this corollary, we observe that we can instantiate the four-round simulatable OT from statistical sender private OT using the work of [MOSV22]. In more detail, in this work, the authors show how to realize four-round simulatable OT by relying, in a black-box way, on what they call a defensible OT. As observed in [MOSV22, Section 3.2 & 3.3] this notion is achieved by a statistical sender private OT, therefore, we obtain a four-round simulatable OT. The list simulatable watchlist protocol can be instantiated using statistical sender private OT as it has been shown in [COSW23a]. For the remaining primitives used in the above protocol, we already observed in Sections 2.6, 3.1, and 4.6 that they can be realized using maliciously secure two-round statistical sender private OT in a black-box way For this corollary to hold, it remains to argue that the protocol uses the mentioned primitives in a black-box way. For the first three rounds, this can be seen directly since the mentioned primitives are all executed on their own. These first three rounds of the protocol also corresponds to the standard [IPS08, IKSS21, IKSS23] protocol with the execution of an additional OT protocol $\mathsf{OT}_{\vec{S},\vec{R}}$. It remains to be argued that the fourth round of the protocol only makes black-box use of the primitives. The critical step here is the conditional disclosure of secrets implemented using the garbled circuit. If the conditional disclosure of secret, i.e., the garbled circuit, needs to evaluate a specific circuit of a cryptographic operation, then this results in a non-black-box use of primitives. In our protocol, we overcome this obstacle by letting the garbled circuit evaluate an information-theoretic operation that does not have any specific dependency on the underlying cryptographic primitive. In more detail, since we required the operations Π_2^{inp} to be information-theoretic (it consists of an operation that outputs a specific subset) our conditional disclosure of secrets uses the inner protocol Π in a black-box way. This concludes the argument for the corollary.

Acknowledgements

Supported in part by DARPA under Cooperative Agreement HR0011-20-2-0025, the Algorand Centers of Excellence programme managed by Algorand Foundation, NSF grants CNS-2246355, CCF-2220450 and CNS-2001096, US-Israel BSF grant 2022370, Amazon Faculty Award, Cisco Research Award and Sunday Group. Any views, opinions, findings, conclusions or recommendations contained herein are those of the author(s) and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, the Department of Defense, the Algorand Foundation, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright annotation therein. The first author has performed part of this work while consulting for Stealth Software Technologies, Inc. The fourth author is supported by an MC2 postdoctoral fellowship.

References

ACJ17. P. Ananth, A. R. Choudhuri, and A. Jain. A new approach to round-optimal secure multiparty computation. In *CRYPTO 2017, Part I*, *LNCS* 10401, pages 468–499. Springer, Heidelberg, August 2017. (Page 3.)

AIK04. B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175. IEEE Computer Society Press, October 2004. (Page 16.)

AIR01. W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT 2001*, *LNCS* 2045, pages 119–135. Springer, Heidelberg, May 2001. (Page 4.)

App17. B. Applebaum. Garbled circuits as randomized encodings of functions: a primer. In *Tutorials on the Foundations of Cryptography*, pages 1–44. Springer International Publishing, 2017. (Page 16.)

BD18. Z. Brakerski and N. Döttling. Two-message statistically sender-private OT from LWE. In *TCC 2018, Part II*, *LNCS* 11240, pages 370–390. Springer, Heidelberg, November 2018. (Pages 4 and 8.)

BGJ$^+$18. S. Badrinarayanan, V. Goyal, A. Jain, Y. T. Kalai, D. Khurana, and A. Sahai. Promise zero knowledge and its applications to round optimal MPC. In *CRYPTO 2018, Part II*, *LNCS* 10992, pages 459–487. Springer, Heidelberg, August 2018. (Pages 3 and 5.)

BHP17. Z. Brakerski, S. Halevi, and A. Polychroniadou. Four round secure computation without setup. In *TCC 2017, Part I*, *LNCS* 10677, pages 645–677. Springer, Heidelberg, November 2017. (Page 3.)

BHR12. M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM CCS 2012*, pages 784–796. ACM Press, October 2012. (Page 16.)

BMR90. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990. (Page 3.)

CCG$^+$20. A. R. Choudhuri, M. Ciampi, V. Goyal, A. Jain, and R. Ostrovsky. Round optimal secure multiparty computation from minimal assumptions. In *TCC 2020, Part II*, *LNCS* 12551, pages 291–319. Springer, Heidelberg, November 2020. (Page 3.)

COSV17. M. Ciampi, R. Ostrovsky, L. Siniscalchi, and I. Visconti. Round-optimal secure two-party computation from trapdoor permutations. In *TCC 2017, Part I*, *LNCS* 10677, pages 678–710. Springer, Heidelberg, November 2017. (Pages 9 and 10.)

COSW23a. M. Ciampi, R. Ostrovsky, L. Siniscalchi, and H. Waldner. List oblivious transfer and applications to round-optimal black-box multiparty coin tossing. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, *Lecture Notes in Computer Science* 14081, pages 459–488. Springer, 2023. (Pages 3, 5, 6, 14, and 67.)

COSW23b. M. Ciampi, R. Ostrovsky, L. Siniscalchi, and H. Waldner. List oblivious transfer and applications to round-optimal black-box multiparty coin tossing. Cryptology ePrint Archive, Paper 2023/1512, 2023. https://eprint.iacr.org/2023/1512. (Page 13.)

COWZ22. M. Ciampi, R. Ostrovsky, H. Waldner, and V. Zikas. Round-optimal and communication-efficient multiparty computation. In *EUROCRYPT 2022, Part I*, *LNCS* 13275, pages 65–95. Springer, Heidelberg, May / June 2022. (Page 3.)

CRSW22. M. Ciampi, D. Ravi, L. Siniscalchi, and H. Waldner. Round-optimal multi-party computation with identifiable abort. In *EUROCRYPT 2022, Part I*, *LNCS* 13275, pages 335–364. Springer, Heidelberg, May / June 2022. (Page 7.)

DGI$^+$19. N. Döttling, S. Garg, Y. Ishai, G. Malavolta, T. Mour, and R. Ostrovsky. Trapdoor hash functions and their applications. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 3–32. Springer, Heidelberg, August 2019. (Page 4.)

DGL$^+$21. X. Duan, V. Goyal, H. Li, R. Ostrovsky, A. Polychroniadou, and Y. Song. ACCO: algebraic computation with comparison. In *CCSW@CCS '21: Proceedings of the 2021 on Cloud Computing Security Workshop, Virtual Event, Republic of Korea, 15 November 2021*, pages 21–38. ACM, 2021. (Page 3.)

GB96. S. Goldwasser and M. Bellare. Lecture notes on cryptography. *Summer course "Cryptography and computer security" at MIT*, 1999:1999, 1996. (Page 17.)

GIS18. S. Garg, Y. Ishai, and A. Srinivasan. Two-round MPC: Information-theoretic and black-box. In *TCC 2018, Part I*, *LNCS* 11239, pages 123–151. Springer, Heidelberg, November 2018. (Pages 34 and 38.)

GLO$^+$21. V. Goyal, H. Li, R. Ostrovsky, A. Polychroniadou, and Y. Song. ATLAS: efficient and scalable MPC in the honest majority setting. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, *Lecture Notes in Computer Science* 12826, pages 244–274. Springer, 2021. (Page 3.)

GMPP16. S. Garg, P. Mukherjee, O. Pandey, and A. Polychroniadou. The exact round complexity of secure computation. In *EUROCRYPT 2016, Part II*, *LNCS* 9666, pages 448–476. Springer, Heidelberg, May 2016. (Page 3.)

GMW87. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, pages 218–229. ACM Press, May 1987. (Page 3.)

Gol04. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004. (Page 18.)

Goy11. V. Goyal. Constant round non-malleable protocols using one way functions. In *43rd ACM STOC*, pages 695–704. ACM Press, June 2011. (Page 3.)

GS18. S. Garg and A. Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *EUROCRYPT 2018, Part II*, *LNCS* 10821, pages 468–499. Springer, Heidelberg, April / May 2018. (Pages 17, 34, 35, and 38.)

HHPV18. S. Halevi, C. Hazay, A. Polychroniadou, and M. Venkitasubramaniam. Round-optimal secure multi-party computation. In *CRYPTO 2018, Part II*, *LNCS* 10992, pages 488–520. Springer, Heidelberg, August 2018. (Page 3.)

HK12. S. Halevi and Y. T. Kalai. Smooth projective hashing and two-message oblivious transfer. *Journal of Cryptology*, 25(1):158–193, January 2012. (Page 4.)

IKOS07. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 21–30, 2007. (Page 4.)

IKOS09. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM Journal on Computing*, 39(3):1121–1152, 2009. (Page 4.)

IKP10. Y. Ishai, E. Kushilevitz, and A. Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO 2010*, *LNCS* 6223, pages 577–594. Springer, Heidelberg, August 2010. (Pages 13, 19, and 54.)

IKSS21. Y. Ishai, D. Khurana, A. Sahai, and A. Srinivasan. On the round complexity of black-box secure MPC. In *CRYPTO 2021, Part II*, *LNCS* 12826, pages 214–243, Virtual Event, August 2021. Springer, Heidelberg. (Pages 3, 4, 5, 6, 11, 18, 35, 54, 62, and 67.)

IKSS23. Y. Ishai, D. Khurana, A. Sahai, and A. Srinivasan. Round-optimal black-box MPC in the plain model. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, *Lecture Notes in Computer Science* 14081, pages 393–426. Springer, 2023. (Pages 3, 4, 6, 31, 32, 34, 35, 37, 45, 49, and 67.)

IPS08. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO 2008*, *LNCS* 5157, pages 572–591. Springer, Heidelberg, August 2008. (Pages 3, 4, 38, and 67.)

Kal05. Y. T. Kalai. Smooth projective hashing and two-message oblivious transfer. In *EUROCRYPT 2005*, *LNCS* 3494, pages 78–95. Springer, Heidelberg, May 2005. (Page 4.)

Kil88. J. Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988. (Pages 3 and 38.)

KO04. J. Katz and R. Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO 2004*, *LNCS* 3152, pages 335–354. Springer, Heidelberg, August 2004. (Page 3.)

KOS03. J. Katz, R. Ostrovsky, and A. Smith. Round efficiency of multi-party computation with a dishonest majority. In *EUROCRYPT 2003*, *LNCS* 2656, pages 578–595. Springer, Heidelberg, May 2003. (Page 3.)

LP09. Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009. (Page 16.)

MOSV22. V. Madathil, C. Orsini, A. Scafuro, and D. Venturi. From privacy-only to simulatable OT: black-box, round-optimal, information-theoretic. In *3rd Conference on Information-Theoretic Cryptography, ITC 2022, July 5-7, 2022, Cambridge, MA, USA*, *LIPIcs* 230, pages 5:1–5:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Pages 11, 66, and 67.)

NP01. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *12th SODA*, pages 448–457. ACM-SIAM, January 2001. (Page 4.)

Ode09. G. Oded. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, USA, 1st edition, 2009. (Page 11.)

ORS15. R. Ostrovsky, S. Richelson, and A. Scafuro. Round-optimal black-box two-party computation. In *CRYPTO 2015, Part II*, *LNCS* 9216, pages 339–358. Springer, Heidelberg, August 2015. (Pages 9 and 10.)

Pas04. R. Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *36th ACM STOC*, pages 232–241. ACM Press, June 2004. (Page 3.)

PS21. A. Patra and A. Srinivasan. Three-round secure multiparty computation from black-box two-round oblivious transfer. In *CRYPTO 2021, Part II*, *LNCS* 12826, pages 185–213, Virtual Event, August 2021. Springer, Heidelberg. (Pages 2, 6, 16, 31, 34, 35, 37, 38, and 45.)

PW10. R. Pass and H. Wee. Constant-round non-malleable commitments from sub-exponential one-way functions. In *EUROCRYPT 2010*, *LNCS* 6110, pages 638–655. Springer, Heidelberg, May / June 2010. (Page 3.)

Wee10. H. Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *51st FOCS*, pages 531–540. IEEE Computer Society Press, October 2010. (Page 3.)

Yao86. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. (Pages 3 and 16.)