Title
Round-Optimal Secure Multiparty Computation from Minimal Assumptions

Author
Srinivasan, Akshayaram

Publication Date
2020

Peer reviewed|Thesis/dissertation

Round-Optimal Secure Multiparty Computation from Minimal Assumptions

by

Akshayaram Srinivasan

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Assistant Professor Sanjam Garg, Chair
Associate Professor Prasad Raghavendra
Professor Kenneth A. Ribet

Spring 2020

Round-Optimal Secure Multiparty Computation from Minimal Assumptions

Abstract

Round-Optimal Secure Multiparty Computation from Minimal Assumptions

by

Akshayaram Srinivasan

Doctor of Philosophy in Computer Science

University of California, Berkeley

Assistant Professor Sanjam Garg, Chair

Secure Multiparty Computation (MPC) allows a set of parties, each having its own private data, to compute a function of their joint data such that the parties only learn the output of this function and everything else about their data is hidden. MPC is a fundamental cryptographic abstraction that captures many other natural cryptographic primitives as special cases. It also has a wide range of practical applications such as conducting secure electronic auctions, delegating sensitive computation to untrusted cloud service providers, secure computation on genomic data and training machine learning models on private data.

A long-standing open problem in this area is to construct MPC protocols that have *optimal* (two-round) round complexity under the *weakest* possible cryptographic hardness assumptions.

We *completely resolve* this problem by showing lower bounds and matching upper bounds on the hardness assumptions required for constructing round-optimal MPC. Specifically, we obtain the following results.

- **Black-Box Separation.** We show that there exists no construction of two-round semi-honest MPC protocol for general functions that makes black-box use of a two-round oblivious transfer. As a corollary, a similar separation holds when starting with any 2-party functionality other than oblivious transfer.
- **Non-Black-Box Construction.** We show that the above negative result can be bypassed by making non-black-box use of a two-round oblivious transfer. Specifically, starting from a semi-honest (resp. malicious) secure two-round oblivious transfer protocol, we construct a two-round MPC protocol against semi-honest (resp. malicious) adversaries.

Since any two-round MPC protocol for general functionalities implies a two-round oblivious transfer as a special case, the above two results show that non-black-box use of oblivious transfer is *necessary and sufficient* to construct round-optimal MPC protocols.

To my parents.

Contents

Contents **ii**

List of Figures **iii**

1 Introduction **1**
1.1 Our Results 2
1.2 Subsequent Work 6
1.3 Open Problems 7
1.4 Bibliographic Notes 8

2 Black-Box Separation **9**
2.1 Overview of the Separation Result 9
2.2 Preliminaries 15
2.3 Main Separation Result 19
2.4 Separating Non-Compact HSS from 2-Round OT 34

3 Non-Black-Box Construction **37**
3.1 Universal Composability Framework 37
3.2 Building Blocks 42
3.3 Conforming Protocols 48
3.4 Two-Round MPC: Semi-Honest Case 50
3.5 Two-Round MPC: Malicious Case 58

Bibliography **67**

List of Figures

2.1 Key Steps in the Proof. PDT denotes publicly decodable transcript, $g(x, y) = x \cdot y$ and $f_\times(x, y, \perp) = g(x, y)$. 11

3.1 The Common Reference String Functionality. 42
3.2 General Functionality. 42
3.3 Two-round Semi-Honest MPC. 53
3.4 The program P. 54
3.5 Two-round Malicious MPC. 60

Acknowledgments

The last five years in Berkeley have been amazing and the main reason for making this time truly remarkable is my advisor Sanjam Garg. Sanjam has been an ideal advisor to me, teaching me everything from selecting good problems to writing a research paper. His infectious enthusiasm about cryptography and immense knowledge on the topic has been a constant source of inspiration. The discussions I have had with him have been instrumental in the way I think about a problem and in shaping me as a researcher. More than an advisor he has been a very dear friend. Someday, I hope to be at least half as good as Sanjam in advising students!

I have been fortunate to closely collaborate with two other exceptional researchers, Vipul Goyal and Yuval Ishai. Vipul has guided me from my undergraduate days and hosted me for a summer in MSR India and a short visit to CMU. He introduced me to the wonderful world of secure multiparty computation and non-malleability during those visits. Yuval hosted me at Technion for the last two summers and I had a great time there. Anyone who has visited Yuval can vouch for his amazing hospitality and I am no different! Yuval introduced me to a plethora of research projects, ranging from leakage-resilience to secure computation. The work that led to the results in Chapter 2 started during my visit to the Technion.

I have also had the pleasure of collaborating with many other amazing researchers, who in many ways have helped me throughout my journey: Benny Applebaum, Saikrishna Badrinarayanan, James Bartusek, Andrej Bogdanov, Zvika Brakerski, Nico Döttling, Divya Gupta, Mohammad Hajiabadi, Dakshita Khurana, Ryan Lehmkuhl, Mohammad Mahmoody, Pasin Manurangsi, Daniel Masny, Peihan Miao, Eric Miles, Pratyush Mishra, Pratyay Mukherjee, Rafail Ostrovsky, Omkant Pandey, C. Pandu Rangan, Sunoo Park, Raluca Ada Popa, Silas Richelson, Amit Sahai, Yifan Song, Prashant Nalini Vasudevan, Mark Zhandry, Wenting Zheng, and Chenzhi Zhu.

The theory group in Berkeley has served as my academic home for the last five years. I have been fortunate to have many wonderful academic and non-academic conversations with the faculty and the students.

Lastly, this dissertation would not have been possible without the love and affection of my Amma and Appa. They are the reason I am here and I owe everything to them.

Chapter 1

Introduction

Cryptography has been traditionally viewed as a mechanism that enables *secure communication* between a set of parties. Early research in cryptography has led to the development of novel primitives such as encryption and digital signatures that have allowed parties to communicate data without compromising both the confidentiality and the integrity of the data. However, a drawback of these primitives is that they can only act on *data at rest.* This roughly means that once we encrypt or digitally sign some data, we can no longer perform any computation on the data and the data becomes immutable. Today, with the advent of cloud computing, it is not sufficient to only have data at rest, and in fact, we need the ability to compute on data while maintaining the confidentiality and the integrity. This is what is enabled by *Secure Multiparty Computation.*

Secure multiparty computation (MPC) allows a set of parties to compute a function on their private inputs such that the parties only learn the output of the function and everything else about their inputs is hidden. This notion was introduced in the seminal works of Yao [Yao82; Yao86] and Goldreich, Micali and Wigderson [GMW87] and has served as a cornerstone of modern cryptography. Over the last thirty years, this field as seen tremendous progress in the theoretic constructions of secure protocols as well as constructing concretely efficient protocols for practical purposes. Moreover, tools and techniques developed in this area have helped resolve several long standing open problems in other areas of cryptography and complexity theory. In spite of this phenomenal progress, several fundamental questions in this area remain open and in this thesis, we resolve one such foundational question.

The round-complexity of MPC. The works of Yao [Yao86] and Goldreich, Micali and Wigderson [GMW87] showed that any efficiently computable multiparty functionality can also be computed securely. After these seminal feasibility results, most of the work in this area have focused on constructing *efficient* protocols for securely computing general functions. By efficient protocols, we refer to those protocols that have a "small" overhead in the resources needed for achieving security when compared to the insecure protocol for the same task. In this thesis, we will consider one of the most important efficiency determining parameters called as *round-complexity.* Round complexity of a protocol roughly corresponds

to the number of back-and-forth interactions between the parties. This is a major efficiency bottleneck if the protocols are run over high-latency networks such as between parties who are geographically far apart. Starting from the seminal work of Beaver, Micali and Rogaway [BMR90], a major line of inquiry in this area is to construct secure protocols that have *optimal round-complexity.*

Prior Work. A folklore result that was formally proven in the work of Halevi, Lindell and Pinkas [HLP11] is that there are certain functions that require at least two-rounds of interaction to be computed securely. The first two-round multiparty computation protocol for general functions was given by Garg et al. [Gar+14] based on the assumption that indistinguishability obfuscation [Bar+01; Gar+13a] exists. Later, Gordon et al. [GLS15] improved the assumptions to witness encryption [Gar+13b]. Mukherjee and Wichs [MW16] gave a two-round protocol based on the learning-with-errors assumption (see also [PS16; BP16]). Boyle et al. [BGI16; BGI17] constructed protocols for a constant number of parties under the Decisional Diffie-Hellman assumption. Garg and Srinivasan gave a two-round MPC protocol based on standard assumptions on bilinear maps [GS17]. We note that all the above positive results either rely on heavy cryptographic machinery or on structured interactability assumptions. On the other hand, the only necessary assumption for constructing round-optimal MPC protocols is the existence of a two-round oblivious transfer [Rab81; EGL85].[1] This brings us to the main question addressed in this thesis:

What is the necessary and sufficient cryptographic assumption for constructing two-round secure computation protocols for general functions?

1.1 Our Results

In this thesis, we give a complete answer to the above mentioned problem. Specifically, in the first part of this thesis (see Chapter 2), we show that black-box access to a two-round oblivious transfer is *insufficient* for constructing a two-round MPC protocol for general functions. In the second part (see Chapter 3), we show that this impossibility result can be bypassed using non-black-box access to a two-round oblivious transfer protocol. Thus, non-black access to a two-round oblivious transfer protocol is necessary and sufficient for constructing round-optimal MPC protocol for general functions. We now give more details of our main results.

[1] An oblivious transfer is a secure protocol for computing the following two-party functionality. One of the parties, called as the receiver has a bit $b \in \{0,1\}$ and the other party, called as the sender has two strings $m_0, m_1 \in \{0,1\}^*$. At the end of the protocol, we require the receiver to learn m_b and the sender does not receive any outputs.

1.1.1 Black-Box Separation

Before we give a detailed account of our impossibility result, let us first give a brief overview of *round-preserving black-box reductions* (RPBB reductions for short) from an n-party functionality f to a p-party functionality g in the semi-honest setting.[2] The notion of RPBB reduction can have two distinct flavors. *Strict* RPBB reduction corresponds to the notion known in the MPC literature as a "(non-interactive) construction of f in the g-hybrid model", namely a protocol that securely realizes f using parallel invocations of an oracle computing g (possibly with different sets of parties) and no further interaction. *Free* RPBB reduction is a relaxed notion that refers to a k-round protocol for f that makes a black-box use of any k-round protocol for g and additional communication over point-to-point channels.[3] In this thesis, we assume $k = 2$ by default. The latter notion more closely resembles the complexity-theoretic notion of a "black-box reduction" (with the round preserving property on top). Note that a strict RPBB reduction implies a free RPBB reduction for any k (and a free RPBB separation implies a strict RPBB separation). This follows from a parallel composition theorem for MPC in the semi-honest model (see, e.g., [Can00; Gol04]).

To illustrate the distinction between the two notions, note that in a free RPBB reduction, party A can, for example, generate two different "first messages" of the g protocol and send both of them to party B, which in turn decides (say, based on its input) to only respond to one of these two messages. In a strict RPBB there is no notion of "first message" and the parties can only feed their inputs into the g functionality and obtain the output. Similarly, in a free RPBB reduction parties can transfer messages and randomness of the g protocol to other parties.[4]

Our first and main result in this part rules out free RPBB reductions of general MPC to OT. That is, general 2-round MPC protocols cannot be based on a 2-round OT protocol in a black-box way. In fact, we show that even 3-party computation of fairly simple functionality called $(2,3) - \mathsf{MULTPlus}$ cannot be realized via black-box use of a 2-round OT protocol. In this functionality, there are three parties, Alice, Bob and Carol. Alice and Bob have inputs $a \in \{0,1\}$ and $b \in \{0,1\}$ respectively and Carol does not have any inputs. At the end of the protocol, we want all the three parties to learn $a \cdot b$.

Theorem 1.1.1 (Main Negative Result) $(2,3) - \mathsf{MULTPlus}$ *cannot be securely realized by a 2-round protocol making a black-box use of 2-round OT protocol.*

This must be contrasted with the result of Yao [Yao86] who showed that securely computing a two-party functionality black-box reduces in a round-preserving way to the computation

[2] We deal with semi-honest adversaries as this makes the negative results stronger.

[3] We consider private channels for our black-box separation as this makes the negative results stronger.

[4] This "transferability" feature is commonly used in applications of commitments and signatures. In the context of OT-based MPC, it can be used to realize "security with identifiable abort" given black-box access to an OT protocol [IOZ14], which is impossible given only access to an OT oracle [IOS12]. Other examples for the distinction between the two types of reductions arise in the contexts of complete functionalities [LOZ18] and OT-combiners [Har+05].

of a 2-round OT protocol.

Theorem 1.1.2 (Black-box 2-round 2PC from OT [Yao86]) *Every 2-party functionality g admits an MPC protocol that only makes parallel calls to an OT oracle and a black-box use of a PRG.*

The $(2,3)-\mathsf{MULTPlus}$ functionality is a special case of an *external-output* 3-party functionality. Formally, let $g(x,y)$ be a 2-party functionality. The *external version* of g, is the 3-party functionality f_g that takes x from Alice, y from Bob and delivers $g(x,y)$ to Alice and Bob, and to Carol who holds no input.[5] Two-round protocols for such functionalities turn out to have interesting properties. Specifically, at the core of our main impossibility result (Theorem 1.1.1) lies the following constructive theorem for external functionalities. (See Theorem 2.3.2 for a more detailed version.)

Theorem 1.1.3 (Conversion Theorem) *Let $g(x,y)$ be a 2-party functionality and let f_g be its 3-party external version. Suppose f_g can be securely realized in 2 rounds (over private channels) by making a black-box use of a 2-round OT protocol. Then, f_g can be securely realized over random inputs (and private channels) given only an access to a random oracle. Moreover, in the resulting protocol Carol sends no message, which implies a 2-party protocol for g over random inputs given only a random oracle.*

Haitner et al. [HOZ13] showed that any 2-party functionality that can be securely realized in the random oracle model over random inputs is "trivial" in the sense that it admits a 2-party protocol over random inputs with security against computationally unbounded adversaries. For functionalities whose input domain size is polynomially bounded, security over random inputs implies security on worst-case inputs. (See Proposition 2.2.2.) For such functionalities, we get the following characterization which strengthens Theorem 1.1.1. (See Corollary 2.3.4 for more details.)

Corollary 1.1.4 *Let g be a 2-party symmetric boolean functionality whose domain size is polynomial in the security parameter. Then the external-output 3-party functionality f_g can be securely realized by a 2-round protocol (over private channels) that makes a black-box use of 2-round OT if and only if g is* trivial *in the sense that it can be realized with perfect security in the plain model.*

A notable example for a *non-trivial* 2-party functionality is the AND functionality [CK91; Kus89].

Corollary 1.1.4 is tight in terms of round complexity. With one additional round (namely, a total of 3 rounds), f_g can be black-box reduced to 2-round OT. (Specifically, one can

[5] In fact, for all of our purposes, an even weaker version suffices. In this relaxed version, all parties are allowed to learn the output (for purposes of privacy), but only Carol is *required* to learn it (for purposes of correctness). Since this leads to a cumbersome definition, we stick to the simpler version described above.

use Theorem 1.1.2 to pass the value of $f(x, y)$ to Alice and Bob in two rounds, and then exploit the additional round to send this value to Carol.) The 2-party completeness of OT (Theorem 1.1.2) also implies that Corollary 1.1.4 holds when the OT functionality is replaced by an arbitrary 2-party functionality $h(x, y)$. Overall, we get a separation between all 2-party functionalities and all external-output functionalities f_g whose underlying g is non-trivial.

1.1.2 Non-Black-Box Construction

In second part of the thesis (Chapter 3), we show that black-box impossibility can be bypassed using a non-black-box access to a 2-round oblivious transfer. In a bit more detail, we give a compiler that takes an arbitrary round MPC protocol (which can in particular be realized using a two-round oblivious transfer) and then compile it to a two-round protocol for the same function. We call such a compiler as a *round-collapsing compiler*. Our main theorem is:

Theorem 1.1.5 (Main Positive Result) *Let $\mathcal{X} \in \{$semi-honest in plain model, malicious in common random/reference sting model$\}$. Assuming the existence of a two-round $\mathcal{X}$-OT protocol, there exists a compiler that transforms any polynomial round, $\mathcal{X}$-MPC protocol into a two-round, $\mathcal{X}$-MPC protocol. The resultant two-round protocol makes non-black-box use of the two-round oblivious transfer.*

Previously, such compilers [Gar+14; GLS15; GS17] were only known under comparatively stronger computational assumptions such as indistinguishability obfuscation [Bar+01; Gar+13a], witness encryption [Gar+13b], or using bilinear maps [BF01; Jou00]. Additionally, two-round MPC protocols assuming the learning-with-errors assumptions were known [MW16; PS16; BP16] in the CRS model satisfying semi-malicious security.[6] We now discuss instantiations of the above compiler with known protocols (with larger round complexity) that yield two-round MPC protocols in various settings under minimal assumptions.

Semi-Honest Case. Plugging in the semi-honest secure MPC protocol by Goldreich, Micali, and Wigderson [GMW87], we get the following result:

Corollary 1.1.6 *Assuming the existence of semi-honest, two-round oblivious transfer in the plain model, there exists a semi-honest, two-round multiparty computation protocol in the plain model.*

[6] Semi-malicious security is a strengthening of the semi-honest security wherein the adversary is allowed to choose its random tape arbitrarily. Ashrov et al. [Ash+12] showed that any protocol satisfying semi-malicious security could be upgraded to one with malicious security additionally using Non-Interactive Zero-Knowledge proofs (NIZKs).

Previously, two-round plain model semi-honest MPC protocols were only known assuming indistinguishability obfuscation [Bar+01; Gar+13a], or witness encryption [Gar+13b] or bilinear maps [GS17]. Thus, using two-round plain model OT [BM89; NP01; AIR01; HK12] based on standard number theoretic assumptions such as CDH or DDH or QR or N^{th} residuocity or based on lattice assumptions such as LWE or LPN, this work yields the first two-round semi-honest MPC protocol for polynomial number of parties in the plain model under the same assumptions.

Malicious Case. Plugging in the maliciously secure MPC protocol by Kilian [Kil88] or by Ishai, Prabhakaran and Sahai [IPS08] based on any oblivious transfer, we get the following corollary:

Corollary 1.1.7 *Assuming the existence of UC secure, two-round oblivious transfer against static, malicious adversaries, there exists a UC secure, two-round multiparty computation protocol against static, malicious adversaries.*

Previously, all known two-round maliciously secure MPC protocols required additional use of non-interactive zero-knowledge proofs. As a special case, using a CDH based two-round OT protocol (e.g., [Döt+20]), this work yields the first two-round malicious MPC protocol in the common random string model under the CDH assumption.

Concurrent Work. In a concurrent work and independent work, Benhamouda and Lin [BL18] also construct two-round secure multiparty computation from two-round oblivious transfer. Their construction against semi-honest adversaries is proven under the minimal assumption that two-round, semi-honest oblivious transfer exists. However, their construction against malicious adversaries additionally requires the existence of non-interactive zero-knowledge proofs. In another concurrent and independent work, Boyle et al. [Boy+18] obtain a construction of two-round multiparty computation based on DDH in the public key infrastructure.

1.2 Subsequent Work

The techniques introduced in our work have led to a number of follow-ups and we give a short account of them. Garg et al. [GMS18] gave a construction of a two-round multiparty computation protocol where the number of oblivious transfers executed in the protocol is independent of the circuit size of the function to be securely computed. Since computing oblivious transfers constitutes the bulk of the computational cost, the above work obtained substantial savings. In another work, Garg et al. [GIS18] gave a construction of a two-round multiparty computation protocol that makes black-box use of a DDH-hard or a QR-hard group assuming a strong form of public key infrastructure. Black-box constructions typically tend to be much more efficient than their non-black-box counterparts.

In another work, Cohen et al. [CGZ20], gave a complete characterization of the two-round MPC protocol with minimal number of broadcasts. The positive results in their paper builds on the construction from Theorem 1.1.5. Building on our work, Benhamouda et al. [Ben+18] gave a construction of a two-round MPC protocol that is secure against an adaptive adversary.[7] Bartusek et al. [Bar+20] used our protocol to construct a two-round secure multiparty computation where the first message can be reused across multiple computations. The works of Badrinarayanan et al. [Bad+18] and Choudari et al. [Cho+19] used our protocols as a building block to construct round-optimal protocols in the plain model.

Our round-collapsing compilers have found applications in resolving long standing open problems in the area of information-theoretic secure multiparty computation in the honest majority setting (i.e.,the number of corruptions is less than $n/2$ where the n is the number of parties). Specifically, the works of [Ana+18; ABT18; GIS18] gave constructions of two-round MPC protocol for general functions in the honest majority setting assuming BB access to a pseudorandom generator. If we restrict the function to be computed to be in logspace, then the work of Applebaum et al. [ABT18] gave a construction that is perfectly secure in the semi-honest setting and for the malicious case, the concurrent works of Applebaum et al. [ABT19] and Ananth et al. [Ana+19] gave statistically secure protocols.

1.3 Open Problems

We now give some problems that are left open from our thesis.

Open Problem 1.3.1 *What is the minimal black-box complete primitive for constructing two-round MPC?*

The works of Boyle et al. [Boy+18] and [GIS18] showed that a simple 4-party functionality is sufficient for constructing black-box two-round MPC protocol and the main question here is that is this primitive necessary?

Open Problem 1.3.2 *Can the communication complexity of the two-round MPC protocols based on oblivious transfer be improved?*

The communication cost of the protocol from Theorem 1.1.5 is a polynomial in the circuit size of the function to be computed whereas the two-round protocols based on Learning with errors [MW16; PS16; BP16] have a communication cost that is independent of the circuit size. Is such a polynomial blow-up in the communication cost necessary if the protocol is based on oblivious transfer?

[7] An adaptive adversary is given the power to corrupt the parties during the course of the protocol execution. All our protocols are secure in the static corruption setting where the corrupted parties are decided before the start of the protocol execution.

Open Problem 1.3.3 *Can two-round MPC protocols be constructed in the CRS model based on black-box access to a group where DDH is hard?*

The work of Garg et al. [GIS18] gave such a construction based on a strong setup assumption. The question here is that can this setup be relaxed, say to assumption the existence of a common reference string?

Open Problem 1.3.4 *Is there a construction of three-round MPC protocols that make black-box use of a two-round oblivious transfer?*

Theorem 1.1.1 rules out two-round MPC protocols making black-box use of two-round oblivious transfer but does not rule out three-round protocols.

1.4 Bibliographic Notes

The black-box separation result is based on a joint work with Benny Applebaum, Zvika Brakerski, Sanjam Garg and Yuval Ishai that appeared in ITCS 2020 [App+20]. The non-black-box construction is based on a joint work with Sanjam Garg that appeared in EUROCRYPT 2018 [GS18].

Chapter 2

Black-Box Separation

In this chapter, we give our main black-box separation result. We start with an informal overview of the separation result in Section 2.1 and give some background on black-box separations in Section 2.2. We give the proof of the main result in Section 2.3 and finally give a corollary to separate Homomorphic Secret Sharing [BGI16] from 2-round OT in Section 2.4.

2.1 Overview of the Separation Result

In this section, we give a high-level overview of our techniques in proving the main result (Theorem 1.1.1). To keep the exposition simple, we restrict ourselves to proving the impossibility result for securely computing external-AND.

External-AND Functionality. Let us denote the three parties by (P_1, P_2, P_3). The private input of P_1 is a bit x, the private input of P_2 is a bit y, and P_3 does not have any private input. The functionality $f_\times$ outputs $x \cdot y$ to all the parties. That is, $f_\times(x, y, \perp) = x \cdot y$.

Main Idea. To prove the impossibility result, we define a set of oracles such that 2-round oblivious transfer exists with respect to these oracles, but there exists no 2-round, semi-honest protocol for securely computing $f_\times$. This is sufficient to rule out a black-box transformation from 2-round oblivious transfer to 2-round, 3-party semi-honest protocols for general functionalities. Below, we describe these oracles (throughout this overview, λ denotes the security parameter):

- OT_1 is a random length tripling function that takes in the receiver's choice bit $b \in \{0,1\}$ and its random tape $r \in \{0,1\}^\lambda$ and outputs the receiver's message otm_1.
- OT_2 is a random length tripling function that takes in the receiver's message otm_1, the sender's inputs $m_0, m_1 \in \{0,1\}$, its random tape $s \in \{0,1\}^\lambda$ and outputs the sender's message otm_2.

- OT_3 is a function that takes the transcript $(\mathsf{otm}_1, \mathsf{otm}_2)$ along with (b, r) as input and outputs m_b if there exists unique (m_0, m_1, s) for which $\mathsf{OT}_1(b, r) = \mathsf{otm}_1$ and $\mathsf{OT}_2(\mathsf{otm}_1, m_0, m_1, s) = \mathsf{otm}_2$. Otherwise, it outputs $\perp$.

As observed by [Har+05], the oracles $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ naturally give rise to a 2-round oblivious transfer protocol. Specifically, letting b, r denote the input/randomness of the receiver, and letting $(m_0, m_1), s$ denote the input/randomness of the sender, the protocol proceeds as follows: The receiver sends $\mathsf{otm}_1 = \mathsf{OT}_1(b, r)$ to the sender, who responds with $\mathsf{otm}_2 = \mathsf{OT}_2(\mathsf{otm}_1, m_0, m_1, s)$, allowing the receiver to output the value $\mathsf{OT}_3(\mathsf{otm}_1, \mathsf{otm}_2, b, r)$.

In this work, we prove that the existence of a 2-round protocol for external-AND with respect to the oracles $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ implies a *two-party* protocol for computing $g(x, y) = x \cdot y$ in the random oracle model. (Note that we start with a three-party protocol for an external functionality, and show a two-party protocol for a related functionality.) The existence of such two-party protocol is known to be impossible [CK91; Kus89; HOZ13; MMP14] and therefore the original protocol can also not exist. This proves Theorem 1.1.3 discussed above, and implies Theorem 1.1.1 as a corollary.

Outline. The above result is proven using a sequence of transformations depicted in Figure 2.1.

Step-1: Publicly Decodable Transcript. Let $\overline{\Pi}$ be a 2-round protocol for securely computing $f_\times$ w.r.t. $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$. We first show that this implies a 2-round, 2-party protocol Π for computing the two-party functionality $g = g(x, y) = x \cdot y$, which has an additional special property – the output is *publicly decodable* from the transcript. More formally, there exists a deterministic algorithm Dec that computes the output of the functionality given the transcript of the two-party protocol. In particular, if there exists a protocol Π that computes g with publicly decodable transcript, then Dec on input $\mathbb{T}$ (which is the transcript of the protocol Π) outputs $g(x, y)$. In terms of security, Π is required to have the standard security properties of a two-party (semi-honest) protocol, i.e., the corrupted party does not learn any information about the other party's input except the output.

To transform a 2-round protocol for $f_\times$ into a 2-round protocol Π for g with publicly decodable transcript, we use a standard player emulation technique. Concretely, we ask P_1 to choose a uniform random tape for P_3 and use this random tape to compute all the messages of P_3. Additionally, P_1 and P_2 forwards all its outgoing messages that are sent to each other as well the messages sent to P_3 in the original protocol. P_1 finally sends this random tape in the second round.

This protocol satisfies public decodability since given the transcript of the protocol (which includes the entire view of P_3 in the original protocol), one can run the output computing algorithm of P_3 to learn $g(x, y)$. Further, the security follows directly from the security of the original protocol when (P_1, P_3) and (P_2, P_3) are corrupted.[1]

[1]It is easy to see that the security of the transformed protocol requires security against collusion of P_1, P_3

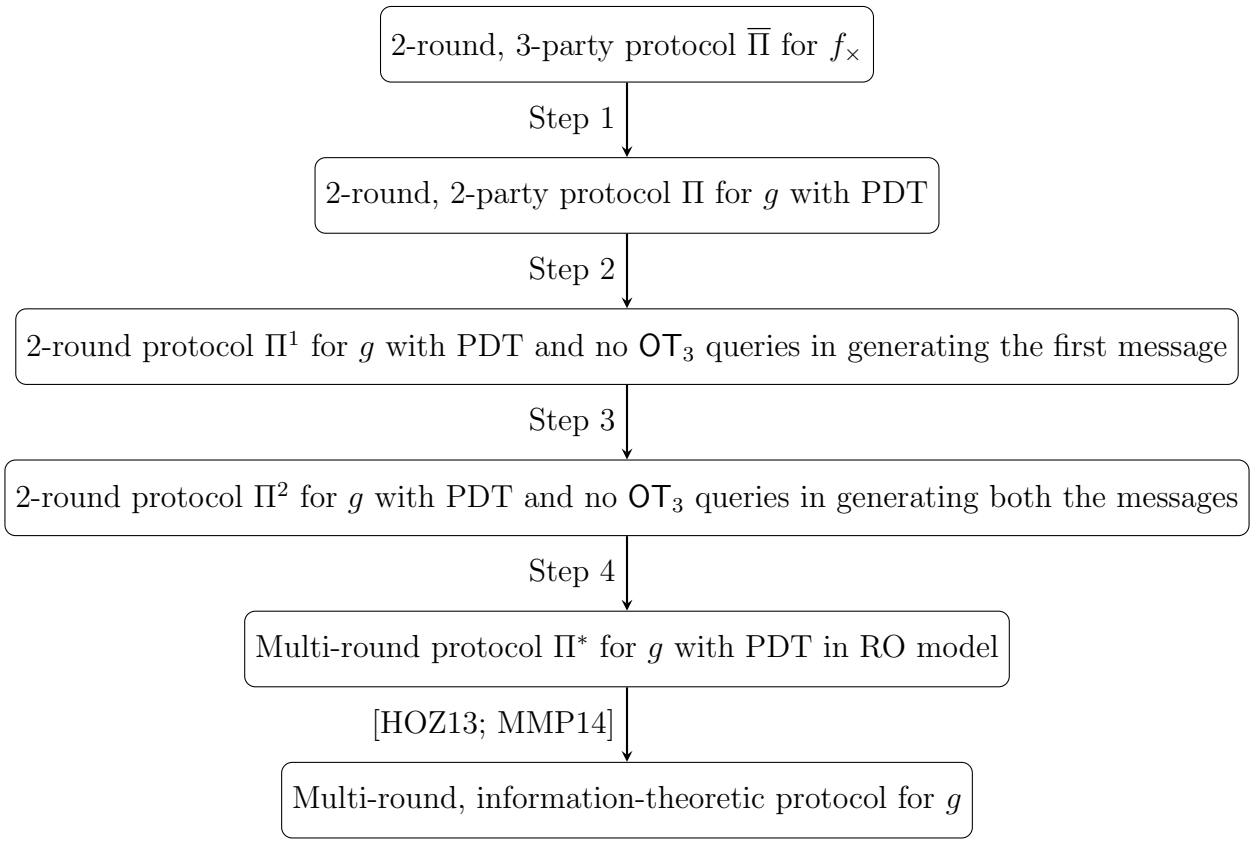

Figure 2.1: Key Steps in the Proof. PDT denotes publicly decodable transcript, $g(x, y) = x \cdot y$ and $f_\times(x, y, \perp) = g(x, y)$.

Remaining Steps – Removing OT_3 Queries. In the remainder of the proof, we show that use of the oracle OT_3 can be removed. More specifically, we show how to convert any two-round two-party secure computation protocol Π with access to $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ and publicly decodable transcript into a two-party protocol that computes the same functionality, but with a few differences. The oracles OT_3 will no longer be used in the new protocol, but this will come at a cost, both in round-complexity and in security:

- The round complexity of the protocol will grow by a polynomial factor (essentially upper bounded by the query complexity of Dec).
- The correctness and security guarantees will only be with respect to random inputs (we call this "security over random inputs"). One instructive way to think about security over random inputs is to think of a protocol between parties that have no

since P_1 has the entire view of P_3. We also require security against (P_2, P_3) collusion since P_1 forwards all its messages sent to P_3 in the second-round of the protocol.

input, and at the beginning of the execution they sample a random input using their local random tape (or shared randomness) and proceed to execute the protocol. Note that this makes simulation easier since we no longer need to worry about consistency with an adversarially chosen (or sampled) input.

In other words, the new protocol Π^* only makes queries to $(\mathsf{OT}_1, \mathsf{OT}_2)$ which are essentially random oracles. Therefore, Π^* securely computes g in the random oracle model. However, it follows from [HOZ13; MMP14] that such a protocol can be used to securely compute g in the information-theoretic setting and this is known to be impossible for the AND functionality [CK91; Kus89] (even with security over random inputs as described above).

The remainder of the overview describes this transformation. We transform Π to Π^* through a sequence of steps. We first transform Π to Π^1 in which the first message function of the protocol does not make any OT_3 queries (Step 2 below). Then, we transform Π^1 to Π^2 such that the first and second message functions of the protocol do not make any OT_3 queries (Step 3 below). Finally, we transform Π^2 to Π^* such that the decoder Dec does not make any OT_3 queries (Step 4 below). It is the final step that incurs the blow-up in the round complexity. Additional details follow.

Step-2: $\Pi \Rightarrow \Pi^1$. The first message function of Π has access to $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ oracles and may make multiple queries to all of them. In order to perform this transformation, we devise a mechanism to emulate the OT_3 oracle without making actual queries to it. Recall that any query to the OT_3 oracle contains $((\mathsf{otm}_1, \mathsf{otm}_2), (b, r))$ and it outputs m_b if and only if there exists (m_0, m_1, s) for which $\mathsf{OT}_1(b, r) = \mathsf{otm}_1$ and $\mathsf{OT}_2(\mathsf{otm}_1, m_0, m_1, s) = \mathsf{otm}_2$. The first step of the OT_3 oracle is easy to emulate; we can query OT_1 on (b, r) and check if the output is otm_1. To emulate the second step, we maintain a list of all the queries/responses made by the first message function to OT_2. If we find an entry $(\mathsf{otm}_1, m_0, m_1, s, \mathsf{otm}_2)$ in this list, we output m_b; else, we output $\perp$. Note that since OT_2 is length tripling, it is injective with overwhelming probability. Thus, if we find such an entry then our emulation is correct. On the other hand, if we don't find such an entry, we output $\perp$ and it can be easily shown that the original oracle also outputs $\perp$ except with negligible probability. Thus, our emulation is statistically close to the real oracle.

Step-3: $\Pi^1 \Rightarrow \Pi^2$**.** It might be tempting to conclude that a similar strategy as before should work even for OT_3 queries made in the second round. That is, maintain the list of queries to the OT_2 oracle and when the second message function makes an OT_3 query, check if there is entry in this list with the response equal to otm_2. If such an entry is found, output the corresponding m_b; else, output $\perp$. This strategy fails because in the second round it is possible that the relevant OT_2 query was made by the other party and therefore it is not possible for each party to only consider the list of OT_2 queries made locally. Note, however, that only one simultanous round of communication has been made by the parties so far. Therefore, it must be the case that the party that made the OT_2 query also made the respective OT_1 query.

To take care of such queries, that we call "correlated queries", we modify the first round of Π^1 as follows. The parties will prepare an additional list L that contains all correlated queries that are "likely" to be asked by the other party. (No OT_3 calls will be made while preparing this list.)

The parties will now send this list L along with the first round message of Π^1. Now, when the second message function of a party in Π^1 attempts to make an OT_3 query on $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$, we first check if otm_1 is valid (by querying OT_1) and then answer this query as follows. If otm_2 is a result of a local query then find the response using the list of local queries/responses. If otm_2 is a correlated query, use the list L sent by the other party to answer. If we don't find any entry in the local list or the correlated list, we output $\perp$. We show that with overwhelming probability, the real oracle also outputs $\perp$ in this case. We also prove that sending this additional list of "likely" correlated queries does not harm the security of Π^2.

To conclude, we describe how the list L is generated, say by P_1. Note that the list needs to be generated at a point where P_1 already decided on its first Π^1 message; now it just needs to come up with L. To this end, P_1 executes many copies of Π^1 executions of P_2, each time with fresh randomness and random input. Then the list L contains the responses to the list of all correlated OT_2 queries, i.e., the valid queries made to OT_3 by "virtual" P_2 such that both OT_1 and OT_2 have been generated by P_1. This will allow to preserve correctness on an average input, and does not violate privacy since given the first Π^1 messages, anyone can sample such executions.

Step-4: $\Pi^2 \Rightarrow \Pi^*$. At the end of step-3, we have a protocol where the first and the second message functions do not make any queries to the OT_3 oracle. However, for the parties to learn the output, they must run the decoder Dec on the transcript, and this decoder might make queries to OT_3. Recall that Dec is a deterministic decoding function whose input is the transcript of the interaction. Further recall that Π^* will be a protocol that does not use OT_3 but will have many communication rounds.

In Π^*, the parties will first execute the two rounds of Π^2 to obtain a transcript. Then one of the parties (say P_1) starts executing the decoder, where for each OT_3 query that the decoder needs to make, if P_2 has made the relevant OT_2 query, then it will "help out" P_1 by sending the decoded value. This will proceed for as many rounds as the number of queries that Dec needs to make, but eventually it will allow P_1 to complete the execution of Dec locally and compute the output of the functionality. We will then need to show that privacy is not harmed in this process. Details follow.

Let us go back to the point where both parties finished executing the two rounds of Π^2 now wish to engage in joint decoding. One of the parties, say P_1, starts running the decoder on the transcript, and along the way maintain the list of $\mathsf{OT}_1, \mathsf{OT}_2$ made by Dec in this process. When the decoder attempts to make an OT_3 query on input $((\mathsf{otm}_1, \mathsf{otm}_2), (b, r))$, P_1 checks if otm_1 is valid (by making a query to OT_1). It then checks if there is an entry $(\mathsf{otm}_1, m_0, m_1, s, \mathsf{otm}_2)$ in the list of OT_2 queries made by the decoder and in the case such

an entry is found, it answers with m_b. If such an entry is not found, P_1 checks its local list of queries/responses made to OT_2 during the generation of the first two messages. If it finds an entry $(\mathsf{otm}_1, m_0, m_1, s, \mathsf{otm}_2)$ in that list, it answers with m_b. If this list does not contain a relevant entry, there are 3 possibilities.

1. otm_2 is not in the image of OT_2 oracle in which case P_1 has to output $\perp$.
2. otm_2 is in the image of OT_2 oracle and P_2 has made this query.
3. otm_2 is in the image of OT_2 oracle and P_2 has not made this query.

The probability that case-3 happens can be shown to be negligible for similar reasons to ones discussed above: if neither party made the relevant OT_2 query then the value otm_2 is almost surely invalid. Thus, P_1 must decide whether it is in case-1 or case-2 and if it is in case-2, it must give the corresponding m_b. To accomplish this, P_1 sends a message to P_2 with (b, otm_2) and asks P_2 to see if there is an entry of the form $(\mathsf{otm}_1, m_0, m_1, s, \mathsf{otm}_2)$ in its local list of queries to OT_2 oracle. If yes, P_2 responds with m_b; else, it responds with $\perp$. P_1 just gives P_2's message as the corresponding response to that query. This blows up the number of rounds of the protocol Π^* proportional to the number of queries made by the decoder.

Observe that Π^* does not make any queries to the OT_3 oracle. At the end, P_1 learns the output $g(x, y)$ and it can send this as the last round message to P_2. Thus, Π^* also has publicly decodable transcript. The correctness of this transformation directly follows since we prove that case-3 happens with negligible probability and if OT_2 is injective (which occurs with overwhelming probability), it follows that if an entry is found in either of the lists of the two parties or on the local list of the decoder, the response given by the emulation is correct.

To see why this transformation is secure, notice that the query $((\mathsf{otm}_1, \mathsf{otm}_2), (b, r))$ is made by the Dec by just looking at the transcript. Hence, there is no harm in P_1 sending (b, otm_2) to the other party. Similarly, if P_2 has indeed made a query to OT_2 such that the response obtained is otm_2, it should follow from the security of Π^2 that the P_2's privacy is not affected if it sends m_b to P_1. Indeed, this information is efficiently learnable given the transcript and an access to the OT_3 oracle. However, there is a subtle issue with this argument which we elaborate next.

Problem of Intersecting Queries. A subtle issue arises when we try to formally reduce the security of Π^* to the security of Π^2. To illustrate this, let us assume the case where P_2 is corrupted. To get a reduction to the security of Π^2, we must give an algorithm that takes the view of P_2 in Π^2 and efficiently generates the view of P_2 in Π^*. In particular, it must generate the additional messages in Π^* given only the view of P_2 in Π^2. This algorithm is allowed to make OT_3 queries as we are trying to give a reduction to the security of Π^2. For the sake of illustration, assume that the Dec makes a single OT_3 query. A natural approach for this algorithm is to take the transcript available in the view of P_2 and start running the

decoder on the transcript. When the decoder makes an OT_3 query, the algorithm uses the real OT_3 oracle to respond to this query. However, notice that the algorithm must generate the messages that correspond to answering this OT_3 query in Π^*. Recall that in Π^*, P_1 first checks in its local list whether there an entry of the form $(\mathsf{otm}_1, m_0, m_1, s, \mathsf{otm}_2)$, and only if such an entry is not found, P_1 sends the message (b, otm_2) to P_2. Thus, to generate the transcript of Π^*, the algorithm must somehow decide whether P_1 would find this entry in its local list or not. However, the algorithm is only given the view of P_2 and does not have any information about the queries that P_1 has made to OT_2.

We see that the problem arises when there is an OT_2 query that potentially was made by both parties. To handle this issue, we resort to the notion of *intersection queries* taken from the key-agreement impossibility result [IR89; BM09]. These works show that it is possible, in polynomial time, to recover a superset of all oracle queries made by both parties (with all but small probability). Given this algorithm, we modify the transformation as follows. The parties will first run the two rounds of communication of Π^2. Then they will run the intersection query finder to recover the intersection query superset. We assume for the purpose of this outline this process is deterministic. Now, upon each potential OT_3 query of the decoder, P_1 will look for the preimage query not only in its query history, but also in the superset of intersection queries, and send a message to P_2 only if the preimage is not found in either of these lists. In particular this means that if the preimage is in the intersection query superset, then we are guaranteed that P_1 will not send a message.

The above modified protocol can be efficiently simulated, since the simulator can also run the intersection query finder and recover the same superset as the parties. Now, if OT_3 gives a valid answer, the simulator looks for a preimage in the intersection query superset. If it finds one, then it concludes that P_1 will not send any message to P_2. If not, then it knows that (except with small probability) exactly one of the parties made the preimage query, and it furthermore knows the internal state of one of the parties, so it knows whether this party made the preimage query. This allows the simulator to always deduce which is the party that made the preimage query and simulate appropriately.

2.2 Preliminaries

Notation. We use λ to denote the security parameter. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be *negligible* if for any positive integer c, there exists λ_0, such that for all $\lambda \geq \lambda_0$, we have $\mu(\lambda) < \lambda^{-c}$. We will use $\mathsf{negl}(\cdot)$ to denote an unspecified negligible function and $\mathsf{poly}(\cdot)$ to denote an unspecified polynomial function. For a string $x \in \{0,1\}^n$ and an index $i \in [n]$, let x_i denote the symbol at the i-th coordinate of x, and for any $T \subseteq [n]$, let $x_T \in \{0,1\}^{|T|}$ denote the projection of x to the coordinates indexed by T. For a function $f : X^n \to Y^n$, we use f_i to denote the function defined as $f(\cdot)_i$ i.e., i-th coordinate of the output, and define f_T analogously.

For a probabilistic algorithm A, we denote by $A(x; r)$ the output of A on input x with the content of the random tape being r. When r is omitted, $A(x)$ denotes a probability

distribution. For a finite set S, we denote by $x \leftarrow S$ the process of sampling x uniformly from the set S. We will use PPT as an abbreviation for Probabilistic Polynomial Time.

2.2.1 Secure Multiparty Computation

Below we define (δ, ε, S)-secure multiparty computation protocols in the presence of oracles with security against semi-honest adversaries. The parameter δ lower-bounds the correctness probability (i.e., the probability that the output is correct), the parameter ε upper-bounds the privacy error, and S upper-bounds the number of oracle queries that a distinguisher is allowed to make. All three parameters are functions of the security parameter λ. The default communication model that we consider in this work assumes that every pair of parties is connected by a private point-to-point channel. This makes negative results stronger. By default, we allow distinguishers as well as honest parties to be unbounded algorithms with a restriction only on the number of oracle queries made. This only strengthens our lower-bounds.

Definition 2.2.1 *Let $\mathcal{O}$ be a set of oracles, $\{X_\lambda\}, \{Y_\lambda\}$ be sequences of finite sets and $f = \{f_\lambda : X_\lambda^n \to Y_\lambda^n\}$ be an n-party functionality. Let $\Pi^{\mathcal{O}}$ be a multiparty protocol computing f by making $\mathsf{poly}(\lambda)$ queries to $\mathcal{O}$. We say that $\Pi^{\mathcal{O}}$ is* efficiently computable *if it can be implemented by oracle-aided Turing machines that run in time $\mathsf{poly}(\lambda)$. We denote the view of party P_i (that includes the input, randomness and message transcript, including the answers of the oracles) in the execution of the protocol $\Pi^{\mathcal{O}}(1^\lambda, x_1, \ldots, x_n)$ (with the input of P_i being equal to x_i) by $\mathsf{view}_{P_i}(1^\lambda, x_1, \ldots, x_n)$ and denote the transcript of the protocol by $\mathbb{T}[\Pi(1^\lambda, x_1, \ldots, x_n)]$. For any $\varepsilon(\lambda), \delta(\lambda) : \mathbb{N} \to [0,1]$ and $S(\lambda) : \mathbb{N} \to \mathbb{N}$, we say that the protocol $\Pi^{\mathcal{O}}$ (δ, ε, S)-securely computes f in the presence of $\mathcal{O} = \{\mathcal{O}_\lambda\}$ if:*

- ***Correctness.*** *For every $\vec{x} = (x_1, \ldots, x_n) \in X_\lambda^n$ and for every $i \in [n]$, the probability that party P_i at the end of protocol $\Pi^{\mathcal{O}}(1^\lambda, x_1, \ldots, x_n)$ outputs the i-th output of $f_\lambda(x_1, \ldots, x_n)$ is at least $\delta(\lambda)$.*
- ***Security.*** *For every set $T \subseteq [n]$ and every pair of inputs $\vec{x} = (x_1, \ldots, x_n)$, $\vec{x}' = (x'_1, \ldots, x'_n)$ for which $f(\vec{x}) = f(\vec{x}')$ and $\vec{x}_T = \vec{x}'_T$, and every non-uniform distinguisher D making at most $S(\lambda)$ queries to the oracle $\mathcal{O}$,*

$$|\Pr[D^{\mathcal{O}}(\{\mathsf{view}_{P_i}(1^\lambda, \vec{x})\}_{i \in T}) = 1] - \Pr[D^{\mathcal{O}}(\{\mathsf{view}_{P_i}(1^\lambda, \vec{x}')\}_{i \in T}) = 1]| \leq \varepsilon(\lambda).$$

We further consider a relaxation of the above to the case where the inputs are selected uniformly from the domain X_λ^n. In this case, the correctness and privacy should hold over a random choice of $(x_1, \ldots, x_n) \leftarrow X_\lambda^n$ and we say that the protocol securely-computes f over random inputs. *Specifically, for every set $T \subseteq [n]$ and for every non-uniform distinguisher D making at most $S(\lambda)$ queries to the oracle $\mathcal{O}$, we require*

$$|\Pr[D^{\mathcal{O}}(\vec{x}, \vec{x}', \{\mathsf{view}_{P_i}(1^\lambda, \vec{x})\}_{i \in T}) = 1] - \Pr[D^{\mathcal{O}}(\vec{x}, \vec{x}', \{\mathsf{view}_{P_i}(1^\lambda, \vec{x}')\}_{i \in T}) = 1]| \leq \varepsilon(\lambda) \tag{2.2.1}$$

where $\vec{x} \leftarrow X_\lambda^n$*, and* $\vec{x}'$ *is sampled uniformly from* X_λ^n *conditioned on* $(\vec{x}_T = \vec{x}'_T$ *and* $f(\vec{x}) = f(\vec{x}'))$*.*

The latter notion of random-input security relaxes standard security analogously to the standard relaxation of semantic security of a cipher to security for *random* plaintext messages. Here too, it is not hard to show that the two notions are equivalent up to a loss which is polynomial in the domain size.

Proposition 2.2.2 (From random-input to standard security) *Let* $\Pi^{\mathcal{O}}$ *be a protocol that* $(1-\alpha, \varepsilon, S)$*-securely computes* f *over random inputs. Let* N *be the size of the domain of* f*. Then the protocol* $\Pi^{\mathcal{O}}$ *is also a* $(1 - N\alpha, N^2\varepsilon, S)$*-secure protocol for* f *(over worst-case inputs).*

Proof If correctness fails over some (worst-case) input $\vec{x} = (x_1, \ldots, x_n) \in X_\lambda^n$ with probability larger than $N\alpha$, then it fails over a random input with probability at least α. Similarly, if a distinguisher D breaks security with advantage $N^2\varepsilon$ over some (worst-case) inputs $\vec{x}$, $\vec{x}'$ for which $f(\vec{x}) = f(\vec{x}')$ and $\vec{x}_T = \vec{x}'_T$, then the there exists a distinguisher D' (with similar complexity) that works over random inputs with advantage at least ε. (Specifically, the distinguisher D' applies D when the inputs are $\vec{x}, \vec{x}'$, and otherwise outputs the constant 1.) ■

Remark 2.2.3 (Simulation-based security) *Our security definitions use indistinguishability between inputs rather than a simulator. However, these definitions (both for standard and for random-input security) imply a standard simulation based definition with respect to a simulator that makes polynomially-many queries to the oracle (but is computationally unbounded). Indeed, the simulator can sample* $\vec{x}'$ *from the uniform distribution on* X_λ^n *conditioned on* $(\vec{x}_T = \vec{x}'_T, f(\vec{x}) = f(\vec{x}'))$ *and run the honest execution of the protocol with the input* $\vec{x}'$*. We note that allowing computationally unbounded simulators can only make negative results stronger. However, we will mainly be interested in functions and distributions for which the above inverse-sampling can be done efficiently. In such cases, computationally unbounded simulation implies the standard notion of efficient simulation.*

Remark 2.2.4 (Notation) *When it is clear from the context, we use* δ, ε*,and* S *to denote* $\delta(\lambda), \varepsilon(\lambda)$*, and* $S(\lambda)$*.*

2.2.2 Black-Box Reductions and Separations

We recall the notion of black-box constructions and separations [RTV04]. We refrain from restating the involved definitional framework that underlies these notions, and we just recall that a cryptographic primitive is characterized by its *correctness* and *security* properties. The security property is defined as a computational task that is attempted by an adversary, where a construction of the primitive is deemed secure if no adversary of a certain class

(usually polynomial time machines) can succeed in this task. We refer to success in the task as "breaking security" of the construction.

We define the notion of fully black-box reductions below. We note that one can consider different flavors of black-box separations, but for this thesis it is sufficient to consider the most basic notion. For the purpose of the definition we assume that the syntax of a cryptographic primitive (and of an adversary) contains only a single algorithm. This does not limit generality (in all cases that are relevant to this work) since we can always replace algorithms $(A_1, \ldots, A_k)$ with a single algorithm A s.t. $A(i, x) = A_i(x)$.

Definition 2.2.5 *A primitive Q reduces to primitive P in a* fully black-box *manner, if there exist polynomial time oracle machines R, B s.t. for every construction p of the primitive P it holds that $q = R^p$ is a construction of the primitive Q with the following properties.*

- *If p has correctness then so does q.*
- *For all A, if A breaks the security of q, then $B^{A,p}$ breaks the security of p.*

We say that R, B is a fully black-box reduction from Q to P.

Definition 2.2.6 *Let P be a cryptographic primitive and let $\mathcal{O}$ be some oracle. A computationally unbounded oracle machine M is $\mathcal{O}$-query-bounded if $M^{\mathcal{O}}$ makes at most a polynomial number of queries to its oracle during its execution. We say that $p = C^{\mathcal{O}}$ is a construction of P in the presence of $\mathcal{O}$ if p has correctness and security against any $\mathcal{O}$-query-bounded adversary.*

The definition extends to the setting where $\mathcal{O}$ is a distribution *over oracles. In such a case the construction $p = C^{\mathcal{O}}$ should be interpreted as a randomized construction, whose first step is to sample the specific oracle from the distribution $\mathcal{O}$, and then to apply C. Likewise, an $\mathcal{O}$-query-bounded adversary is also a randomized entity where in the first step the oracle is sampled and then the machine makes queries as needed.*

The following proposition as an immediate consequence of the definition.

Proposition 2.2.7 *Let Q, P be cryptographic primitives such that Q reduces to P in a fully black-box manner and let $\mathcal{O}$ be an oracle or a distribution over oracles. If there is a construction of P in the presence of $\mathcal{O}$ then there is a construction of Q in the presence of $\mathcal{O}$.*

Proof Let R, B be the fully black-box reduction (see Definition 2.2.5) and let C be such that $p = C^{\mathcal{O}}$ is a construction of P in the presence of $\mathcal{O}$. Define D as R^C, i.e. $q = D^{\mathcal{O}} = R^{C^{\mathcal{O}}}$. From the properties of the reduction, q has correctness. Assume towards contradiction that there exists $\mathcal{O}$-query-bounded A such that $A^{\mathcal{O}}$ breaks the security of q. From the properties of the reduction $B^{A^{\mathcal{O}}}$ would break the security of $p = C^{\mathcal{O}}$. However, $B^{A^{\mathcal{O}}, C^{\mathcal{O}}}$ is $\mathcal{O}$-query-bounded in contradiction to the unconditional security of p against such adversaries. ■

2.3 Main Separation Result

In this section, we state and prove our main result.

3-party Functionality. Let $\{X_\lambda\}_\lambda$, $\{Y_\lambda\}_\lambda$, and $\{Z_\lambda\}_\lambda$ be a sequence of finite sets. Let P_1, P_2, and P_3 to be the three parties. The private input of P_1 is a string $x \in X_\lambda$ and the private input of P_2 is a string $y \in Y_\lambda$. P_3 does not have any private inputs. For any $g_\lambda : X_\lambda \times Y_\lambda \to Z_\lambda$, define a 3-party functionality f_{g_λ} that outputs $g_\lambda(x, y)$ to all the parties. In other words, $f_{g_\lambda}(x, y, \perp) = g_\lambda(x, y)$. We sometimes identify g with the two-party symmetric functionality that takes x from P_1, takes y from P_2 and delivers the output $g(x, y)$ to both parties. For ease of notation, we will drop the subscript λ when denoting f and g.

Lemma 2.3.1 *There exists a set of oracles $\mathcal{O}$ such that:*

- *There exists a 2-round, $(1-\mathsf{negl}(\lambda), \mathsf{negl}(\lambda), \lambda^{\omega(1)})$-secure efficiently-computable protocol for computing any two-party functionality $h : X_\lambda \times Y_\lambda \to Z_\lambda$ in the presence of $\mathcal{O}$.*
- *For any $g : X_\lambda \times Y_\lambda \to Z_\lambda$, if there exists a 2-round, $(1-\mathsf{negl}(\lambda), \mathsf{negl}(\lambda), \lambda^{\omega(1)})$-protocol that securely computes f_g over private channels in the presence of $\mathcal{O}$, then for every polynomial α, there exists a multi-round, $(1 - 1/\alpha(\lambda), 1/\alpha(\lambda), \lambda^{\omega(1)})$-secure protocol that computes f_g on random inputs over private channels in the random oracle model. Moreover, in this protocol, party P_3 does not send any messages and hence, we get a two-party protocol.*

The proof of Lemma 2.3.1 is postponed to the next subsection. We continue by exploring the implications of the lemma. The conversion theorem from Chapter 1 (Theorem 1.1.3), can now be formally derived.

Theorem 2.3.2 (Conversion Theorem, Theorem 1.1.3, restated) *Let $g(x, y)$ be a 2-party functionality. The external version of g is the 3-party functionality f_g that takes x from Alice, y from Bob and delivers $g(x, y)$ to Alice and Bob, and to Carol who holds no input.*

Suppose there is a fully-black-box reduction from 2-round secure computation of f_g to 2-round OT over private channels. Then, for every polynomial α, the functionality f_g can be $(1 - 1/\alpha(\lambda), 1/\alpha(\lambda), \lambda^{\omega(1)})$-securely computed over random inputs given only an access to a random oracle over private channels. Moreover, in the resulting protocol Carol sends no message and so it yields a two-party protocol for $(1 - 1/\alpha(\lambda), 1/\alpha(\lambda), \lambda^{\omega(1)})$-securely computing g with over random inputs given only an access to a random oracle.

Proof We apply Proposition 2.2.7 where the primitive P is set to be 2-round OT and the primitive Q is set to be 2-round secure computation of f_g. By the first part of Lemma 2.3.1, there is a construction of P in the presence of $\mathcal{O}$. Therefore by the proposition, there is also

a construction of 2-round secure computation of f_g in the presence of $\mathcal{O}$. We now apply the second part of Lemma 2.3.1, and derive the theorem. ■

To simplify the following statements, let us say that a functionality f can be *weakly-realized* at the presence of oracle $\mathcal{O}$ over worst-case inputs (resp., random inputs) if for every polynomial $\alpha(\cdot)$ there exists a protocol that $(1 - 1/\alpha(\lambda), 1/\alpha(\lambda), \lambda^{\omega(1)})$-securely computes f over private channels on worst-case inputs (resp., random inputs) given an access to the oracle $\mathcal{O}$. (That is, it achieves privacy and correctness errors of $1/\alpha(\lambda)$ against super-polynomial adversaries.)

The following proposition follows immediately from the transference theorem of Haitner et al. [HOZ13, Theorem 1.1].

Proposition 2.3.3 (RO removal) *If a 2-party functionality g can be weakly-realized over random inputs given an access to a random oracle, then it can also be weakly-realized over random inputs in the plain model.*

Let us restrict our attention to 2-party (symmetric) Boolean functionalities g whose domain is polynomially bounded in the security parameter. By Proposition 2.2.2, if g can be weakly-realized over random inputs then it can be weakly-realized over worst-case inputs. We use this to show that f_g has a 2-round black-box reduction to a 2-round OT if and only if g is *trivial*. Here we call a 2-party functionality *trivial* if it can be weakly-realized over worst-case inputs in the plain model. More precisely, it is trivial if it admits a 2-party information-theoretic semi-honest protocol over worst-case inputs with privacy and correctness errors of $1/\alpha(\lambda)$ for an arbitrary, predefined, polynomial α.

Corollary 2.3.4 (Corollary 1.1.4 restated) *Let f_g be an external-output version of a Boolean 2-party functionality g, whose domain is polynomially-bounded. Then f_g can be securely realized by a 2-round private-channel protocol that makes a black-box use of 2-round OT if and only if g is trivial. Specifically, the two-party* AND *functionality that takes in $x, y \in \{0,1\}$ and outputs $x \cdot y$ is non-trivial and therefore it cannot be computed by a 2-round protocol that makes a black-box use of 2-round OT.*

Proof Suppose that f_g can be securely-computed by a 2-round private-channel protocol that makes a black-box use of 2-round OT. Then, by Proposition 2.3.3 and Theorem 2.3.2, the functionality g can be weakly-realized in the plain model with information-theoretic security over random inputs. Since the domain of g is polynomially-bounded, Proposition 2.2.2 further implies that f_g can be weakly-realized in the plain model with information-theoretic security over *worst-case inputs* and therefore it is trivial.

For the other direction, the classical impossibility result of Chor and Kushilevitz [CK91] shows that a 2-party Boolean function g is trivial if and only if it can be written as $g(x, y) = g_1(x) \oplus g_2(y)$. For such a trivial g, the function f_g admits a 2-round perfect protocol in the plain model; In the first round, P_1 sends a random pad r to P_2, and in the second round P_1 (resp. P_2) sends to the other two parties the value $g_1(x) \oplus r$ (resp., $g_2(y) \oplus r$). Consequently,

f_g can be securely-computed by a 2-round private-channel protocol that makes a black-box use of 2-round OT. Finally, the non-triviality of AND follows from [BGW88; CK91].

■

2.3.1 Proof of Lemma 2.3.1

We start by describing the oracles.

Oracles $\mathcal{O}$. $\mathcal{O}$ is a triple $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3) = \{(\mathsf{OT}_1^n, \mathsf{OT}_2^n, \mathsf{OT}_3^n)\}_{n\in\mathbb{N}}$ with the following syntax (We will use OT_1 to denote $\{\mathsf{OT}_1^n\}_{n\in\mathbb{N}}$ and so on).

- OT_1^n is a random length tripling function that takes in the receiver's choice bit $b \in \{0,1\}$ and its random tape $r \in \{0,1\}^n$ and outputs the receiver's message otm_1.
- OT_2^n is a random length tripling function that takes in the receiver's message otm_1, the sender's inputs $m_0, m_1 \in \{0,1\}$, its random tape $s \in \{0,1\}^n$ and outputs the sender's message otm_2.
- OT_3^n is a function that takes the transcript $(\mathsf{otm}_1, \mathsf{otm}_2)$ along with (b, r) as input and outputs m_b if there exists a unique (m_0, m_1, s) for which $\mathsf{OT}_1(b, r) = \mathsf{otm}_1$ and $\mathsf{OT}_2(\mathsf{otm}_1, m_0, m_1, s) = \mathsf{otm}_2$. Otherwise, it outputs $\perp$.

Notation. The response obtained when $(b, r) \in \{0,1\} \times \{0,1\}^*$ (resp. $(\mathsf{otm}_1, m_0, m_1, s) \in \{0,1\}^* \times \{0,1\} \times \{0,1\} \times \{0,1\}^*$) is queried to OT_1 (resp. OT_2) is given by $\mathsf{OT}_1^{|r|}(b, r)$ (resp. $\mathsf{OT}_2^{|s|}(\mathsf{otm}_1, m_0, m_1, s)$ if $|\mathsf{otm}_1| = 3|s| + 3$; else $\perp$.).

We now recall some basic properties of these oracles.

Fact 2.3.5 *With probability at least $1 - 2^{-n}$, the oracles OT_1^n and OT_2^n are injective.*

Fact 2.3.6 *Let $Q = \{(x_i, y_i)\}_{i\in[q]}$ denote a set of query/response pairs where $x_i \in \{0,1\}^n$ and $y_i \in \{0,1\}^{3n}$ for every $i \in [q]$. Let H_Q denote the random variable that represents a random oracle $H : \{0,1\}^n \to \{0,1\}^{3n}$ conditioned on being consistent with Q, i.e., $H(x_i) = y_i$ for every $i \in [q]$. Then, for any $y \in \{0,1\}^{3n}$, $y \neq y_i$ for all $i \in [q]$, the probability that y is in the image of H_Q is at most 2^{-2n}.*

As already mentioned, the oracles $\mathcal{O}$ naturally give rise to a two-round oblivious transfer protocol [Har+05], and, by Yao's completeness result (Theorem 1.1.2), to a general two-round protocol for two-party functionalities.

Proposition 2.3.7 ([Har+05; Yao86]) *An efficiently computable 2-round oblivious transfer protocol with negligible security and correctness errors exists in the presence of $\mathcal{O}$. Consequently, any two party functionality $h : X_\lambda \times Y_\lambda \to Z_\lambda$ can be efficiently computed by a 2-round, $(1 - \mathsf{negl}(\lambda), \mathsf{negl}(\lambda), \lambda^{\omega(1)})$-secure protocol in the presence of $\mathcal{O}$.*

The following lemma (whose proof is postponed to the next subsection) will complete the proof of Lemma 2.3.1.

Lemma 2.3.8 *Suppose there exists a 2-round,* $(1 - \mathsf{negl}(\lambda), \mathsf{negl}(\lambda), \lambda^{\omega(1)})$*-protocol* $\overline{\Pi}$ *that securely computes* f_g *over private-channels in the presence of* $\mathcal{O}$*. Then, for every polynomial* α*, there exists a multi-round,* $(1-1/\alpha(\lambda), 1/\alpha(\lambda), \lambda^{\omega(1)})$*-secure protocol* Π^* *that computes* f_g *on random inputs over private-channels in the random oracle model. Furthermore,* P_3 *sends no messages in this protocol and hence, we get a two-party protocol for computing* g*.*

2.3.2 Proof of Lemma 2.3.8

Following the outline sketched in Section 2.1, we gradually transform (in each of the following subsections) the protocol in the hypothesis of Lemma 2.3.8 to the protocol in the implication.

2.3.2.1 Publicly-Decodable Protocol

We switch terminology and move from three-party protocols in which the third party is silent (as in Lemma 2.3.8) to the, more convenient terminology of two-party protocols with *publicly-decodable transcripts*.

Definition 2.3.9 *A two-party protocol is* publicly decodable *if at the final step* P_1 *and* P_2 *compute their output by applying a deterministic algorithm* Dec *on the transcript.*

In general any protocol can be transformed into a publicly decodable protocol at the expense of adding an additional message (in which P_1 sends its output). In the following lemma, we show that a three-party protocol for f_g can be transformed into a two-party publicly-decodable protocol for g without any overhead in the round complexity.

Lemma 2.3.10 *Suppose there exists a 2-round,* (δ, ε, S)*-secure protocol* $\overline{\Pi}$ *for* f_g *in the presence of oracle* $\mathcal{O}$ *that makes* $\mathsf{poly}(\lambda)$ *queries to* $\mathcal{O}$*. Then there exists a publicly-decodable* $(\delta, \varepsilon, \Omega(S))$*-secure protocol* Π *for computing* g *in two rounds in the presence of* $\mathcal{O}$ *making* $\mathsf{poly}(\lambda)$ *oracle queries.*

Proof The transformation is via standard player emulation technique.We now describe Π.

1. In round-1, party P_1 will choose an uniform random tape r_3 for P_3 and emulate P_3 using this random tape. P_1 generates the first round messages $\mathsf{msg}^1_{3\to2}, \mathsf{msg}^1_{3\to1}$ sent by P_3 in $\overline{\Pi}$ to P_2, P_1 respectively using r_3. P_1 also generates its first round messages $\mathsf{msg}^1_{1\to2}$ and $\mathsf{msg}^1_{1\to3}$ to be sent to P_2 and P_3 respectively in $\overline{\Pi}$. It sends $(\mathsf{msg}^1_{1\to2}, \mathsf{msg}^1_{3\to2})$ to P_2 in the first round of Π. P_2 generates $(\mathsf{msg}^1_{2\to1}, \mathsf{msg}^1_{2\to3})$ which are the first round messages to be sent to P_1 and P_3 respectively in $\overline{\Pi}$ and sends them to P_1.

2. In round-2, P_1 generates $\mathsf{msg}^2_{1\to 2}$ and $\mathsf{msg}^2_{1\to 3}$ using its private input, its randomness and the messages it received so far. It also uses r_3 and the messages intended to P_3 to generate P_3's round-2 message $\mathsf{msg}^2_{3\to 2}$ to P_2 in $\overline{\Pi}$. It sends $(r_3, \mathsf{msg}^1_{1\to 3}, \mathsf{msg}^2_{1\to 3}, \mathsf{msg}^2_{1\to 2}, \mathsf{msg}^2_{3\to 2})$ to P_2. Similarly, in round-2, P_2 sends $(\mathsf{msg}^2_{2\to 1}, \mathsf{msg}^2_{2\to 3})$ to P_1.

Finally, P_1 and P_2 compute the decoding algorithm Dec executing the output computing algorithm of P_3 using $r_3, \mathsf{msg}^1_{1\to 3}, \mathsf{msg}^2_{1\to 3}, \mathsf{msg}^1_{2\to 3}, \mathsf{msg}^2_{2\to 3}$ as input to output $g(x, y)$.

For every subset $T \subseteq [3]$, let $\overline{\mathsf{Sim}}^{\mathcal{O}}_T$ be the simulator for $\overline{\Pi}$ when the subset T gets corrupted. We set $\mathsf{Sim}^{\mathcal{O}}_1 = \overline{\mathsf{Sim}}^{\mathcal{O}}_{\{1,3\}}$, $\mathsf{Sim}^{\mathcal{O}}_2 = \overline{\mathsf{Sim}}^{\mathcal{O}}_{\{2,3\}}$ and the security and correctness properties follow directly from the guarantees of $\overline{\Pi}$. ■

Next steps. By Lemma 2.3.10, the hypothesis of Lemma 2.3.8 implies a 2-round (δ, ε, S)-secure publicly-decodable protocol Π for g in the presence of oracle $\mathcal{O}$ that makes $\mathsf{poly}(\lambda)$ queries to $\mathcal{O}$ where $\delta = 1 - \mathsf{negl}(\lambda)$, $\varepsilon = \mathsf{negl}(\lambda)$ and $S = \lambda^{\omega(1)}$). For a polynomial $\rho(\lambda)$, our goal is to construct a new a multi-round, $(\delta - O(1/\rho(\lambda)), \varepsilon - O(1/\rho(\lambda)), S - \mathsf{poly}(\rho, \lambda))$-secure two-party protocol Π^* that computes g over random inputs. This will be done via a sequence of transformations. In the first transformation, we remove all the queries made to the OT_3 oracle in generating the first round message of the protocol. In the second transformation, we remove all the OT_3 oracle queries during the generation of the second round message of the protocol. In the final transformation, we remove the OT_3 oracle queries made by Dec.

2.3.2.2 Transformation-1: $\Pi \Rightarrow \Pi^1$

We now transform Π to Π^1 such that no OT_3 queries are made during the generation of the first round message in Π^1. We parameterize Π^1 with an arbitrary polynomial ρ (which will determine the correctness and the security error) and denote by $\Pi^1[\rho]$ the parameterized version. We now give the description of Π^1 below.

Transformation $\Pi \Rightarrow \Pi^1$:

- **Parameters:** Let $q = q(\lambda)$ be the number of oracle queries that each party makes in the first and the second rounds of Π. Let $\rho = \rho(\lambda)$ be an arbitrary polynomial and set $p = \frac{1}{(q+1)\cdot\rho}$.
- **Preprocessing Phase:** For every $\lambda' \leq \log(1/p)$, each party $P \in \{P_1, P_2\}$ makes oracle queries to $\mathsf{OT}^{\lambda'}_2$ on every point in the domain. Party P creates a list L^P_2 with the queries/responses to the OT_2 oracle respectively.
- **Round-1:** In round-1 of Π^1, party P starts executing the first round message function of Π (on their private input and randomness) and appends the list L^P_2 with the queries/responses made to the OT_2 oracle in the generation of the first round message. Whenever the first message function of Π makes an OT_3 oracle query on $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$, the P generates the response to this query as follows:

- It makes an OT_1 oracle call on (b, r) and checks if the response is otm_1. If not, it outputs $\perp$ as the response to the OT_3 query.
- If $|r| \leq \log(1/p)$, this query can be answered by an exhaustive search on the list L_2^P as it contains all the responses to every point in the domain of OT_2.
- Otherwise, P checks if there is an entry $((\mathsf{otm}_1, m_0, m_1, s), \mathsf{otm}_2)$ in the list L_2^P. If such an entry is not found, it outputs $\perp$ as the response to the corresponding OT_3 query. Else, it outputs m_b.

- **Round-2 and Decoder:** Remain as in Π.

Claim 2.3.11 *Let ρ be an arbitrary polynomial. Then, $\Pi^1[\rho]$ is a publicly-decodable protocol that computes g in 2-rounds with $(\delta - 1/\rho(\lambda), \varepsilon + 1/\rho(\lambda), S)$-security in the presence of $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ making $\mathsf{poly}(\lambda)$ oracle queries.*

Proof We now argue that the party's emulation of OT_3 oracle in the first round is $1/\rho(\lambda)$-close to the real oracle. Notice that if $\mathsf{OT}_1(b, r) \neq \mathsf{otm}_1$ or if $|r| \leq \log(1/p)$, then the emulation is perfect. We now show that if $r > \log(1/p)$, the emulation is $1/\rho(\lambda)$-close to the real oracle. To see this, observe that for any query such that otm_2 is found in the list L_P^2, the emulation of the oracle is perfect as long as OT_2 is injective and this happens with probability at least $1 - p$ (Fact 2.3.5). If such a response is not found then the probability (over the choice of the random oracle OT_2) that a string otm_2 is in the image of this oracle is at most p (Fact 2.3.6). Thus, in this case, both the real oracle and party's emulation will output $\perp$ except with probability p and hence, by a standard union bound, party's emulation of OT_3 oracle is $(q+1)p = 1/\rho(\lambda)$-close to the real oracle.

To argue security, notice that with all but $(q+1)p$ probability over random choice of OT_2, party's emulation of the OT_3 oracle in Π^1 is identical to the real oracle and hence, for every inputs x, y and every fixing of the randomness of the parties, the view of each party in Π^1 will be $(q+1)p$-close to their view in Π (where the probability is taken over the choice of OT_2). The security follows directly from the security of Π. ■

2.3.2.3 Transformation-2: $\Pi^1 \Rightarrow \Pi^2$

We now transform Π^1 to Π^2 such that in protocol Π^2, the parties do not make any OT_3 oracle queries when generating the second round message. As mentioned in Section 2.1, in this transformation each party P finds a list, L^P, of "likely" correlated queries that are made by the other party $\overline{P}$ with sufficiently large probability when the inputs of $\overline{P}$ are chosen at random. We give the description of this transformation below.

Transformation $\Pi^1 \Rightarrow \Pi^2$:

- **Parameters:** Let $\rho = \rho(\lambda)$ be an arbitrary polynomial. Let $q_1 = q_1(\lambda)$ be the number of oracle queries that each party makes in the first and the second rounds of Π^1. We define $t = q_1 \lambda \rho$ and set $p = \frac{1}{t q_1 \rho}$.

- **Preprocessing Phase:** For every $\lambda' \leq \log(1/p)$, each party $P \in \{P_1, P_2\}$ makes oracle queries to $\mathsf{OT}_2^{\lambda'}$ on every point in the domain. Party P creates a list L_2^P with the queries/responses to the OT_2 oracle.

- **Round-1:** P_1 (resp., P_2) has input x (resp., y) and random tape R_1 (resp., R_2) for the protocol $\Pi^1[\rho]$. (Additional random bits will be sampled on the fly.) The first round message from party $P \in \{P_1, \mathsf{P}_2\}$ to the other party $\overline{P}$ is generated as follows:

  1. P runs the first round message function of $\Pi^1[\rho]$, and whenever she accesses the oracle OT_2, she appends the query/response pair to the list L_2^P. Let $\pi^1_{1,P\to\overline{P}}$ denote the first-round message (of Π^1) that is generated in this process. Here, we view the preprocessing part of Π^1 as part of the first message function.
  2. Next, P initializes an empty list L^P and runs t "auxiliary" executions of the other party $\overline{P}$ in Π^1 as follows.
  3. In every execution i, P samples a random input and a random tape for $\overline{P}$ and computes its first message. This part of the computation may call the oracles OT_1 and OT_2, and P maintains the corresponding list ℓ_2^i of query/response pairs to the oracle OT_2. Next, P calls the second-message function of $\overline{P}$ in Π^1 (on the same input/random tape) and feeds $\pi^1_{1,P\to\overline{P}}$ as the first message of P in the emulated protocol. During this computation, calls to the oracle OT_2 are recorded in the list ℓ_2^i as before. In addition, any OT_3-query $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ is emulated as follows:
     - Make an OT_1 oracle call on (b, r) and checks if the response is otm_1. If not, output $\perp$ as the response to the OT_3 query.
     - If $|r| \leq \log(1/p)$, this query can be answered by an exhaustive search on the list L_2^P as it contains all the responses to every point in the domain of OT_2.
     - Otherwise, check if there is an entry $((\mathsf{otm}_1, m_0, m_1, s), \mathsf{otm}_2)$ in the list, ℓ_2^i, of queries/responses to the OT_2 oracle made by the emulated $\overline{P}$ in that specific execution. If yes, output m_b.
     - Else, in list L_2^P, check if there is an entry $((\mathsf{otm}_1, m_0, m_1, s), \mathsf{otm}_2)$. If yes, output m_b and add $((\mathsf{otm}_1, \mathsf{otm}_2, (b, r)), m_b)$ to the list L^P. Otherwise, output $\perp$.
  4. P sends $\pi^1_{1,P\to\overline{P}}$ and the list L^P to the other party.

- **Round-2:** The second round message from party $P \in \{P_1, \mathsf{P}_2\}$ to $\overline{P}$ is generated as follows:

  1. P starts executing the second message function of $\Pi^1[\rho]$ with the first round message from $\overline{P}$ set to $\pi^1_{1,\overline{P}\to P}$ and extends the list L_2^P with those queries made by the second message function. Now, it emulates the access to the oracle OT_3 on a query $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ as follows:

a) It makes an OT_1 oracle call on (b, r) and checks if the response is otm_1. If not, it outputs $\perp$ as the response to the OT_3 query.

b) If $|r| \leq \log(1/p)$, this query can be answered by an exhaustive search on the list L_2^P as it contains all the responses to every point in the domain of OT_2.

c) Else, it checks if there is an entry $((\mathsf{otm}_1, m_0, m_1, s), \mathsf{otm}_2)$ in the list L_2^P. If such an entry is found, it outputs m_b.

d) Else, it uses the list $L^{\overline{P}}$ obtained from the other party and checks if there is an entry $((\mathsf{otm}_1, \mathsf{otm}_2, (b, r)), m_b)$. If yes, it outputs m_b. Else, it outputs $\perp$.

- **Decoding:** Both parties take the transcript, remove the lists (L^1, L^2) sent by the parties, and apply the decoder Dec of Π.

Remark 2.3.12 *It is instructive to note that if the input space is large (say larger than n) then the list $L^{\overline{P}}$ may completely miss a query that happens with high probability on a* specific input *of $\overline{P}$. For this reason, the transformation achieves correctness (and security) only with respect to random inputs.*

Claim 2.3.13 *Let ρ be an arbitrary polynomial. Then $\Pi^2[\rho]$ is a 2-round publicly-decodable protocol computing g on random inputs with $(\delta - O(1/\rho(\lambda)), \varepsilon + O(1/\rho(\lambda)), S - \mathsf{poly}(\lambda, \rho))$-security in the presence of $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ making $\mathsf{poly}(\lambda, \rho)$ oracle queries.*

Proof Let G (for good) denote the event in which Π^2's emulation of the OT_3 oracle to the second message of function of Π^1 is identical to the real oracle. To prove correctness, it suffices to show that G happens except with $O(1/\rho(\lambda))$ where the probability is taken over the inputs (x, y), the random tapes, and the oracles.

As before, if $\mathsf{OT}_1(b, r) \neq \mathsf{otm}_1$ or if $|r| \leq \log(1/p)$, the emulation is perfect. From now on we therefore consider only OT_3 queries whose r part is longer than $\log(1/p)$. Let I denote the event that the oracles OT_1 and OT_2 are injective, which happens with probability at least $1 - p$ by Fact 2.3.5. Under I, every query $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ for which otm_2 is found in the list of queries/responses to the OT_2 oracle or in the list L sent by the other party, is being emulated perfectly.

It suffices to deal with the case that the response is not found in any of the lists available to P, and so the emulation outputs $\perp$ (in Step 1d). Let G_1 (resp., G_2) denote the event in which every such $\perp$-query, made by P_1 (resp., P_2), is also evaluated to $\perp$ by the oracle OT_3. We show that G_1 fails to happens with probability at most $q_1(p + 1/(\rho q_1) + e^{-\lambda})$. (A symmetric argument applies to G_2 as well.)

For every query $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ for which P_1 reaches to Step 1d, we distinguish between two cases. If otm_2 was not a response obtained by one of the queries made by one of the parties in Π^1, then the probability (over the choice of the oracles) that the OT_3 oracle does not output $\perp$ is at most p (by Fact 2.3.6). Otherwise, we have the following event ($\star$): The OT_3-query $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ was issued by P_1 on a string otm_2 that was obtained by P_2 and the response to this query does not appear in the list L^{P_2} that P_2 sent in the first round

of Π^2. Call such a query $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ "light" if it is issued by the second-message function of P_1 in $\Pi^1[\rho]$ with probability at most $1/(q_1\rho)$, and "heavy" otherwise. Here the probability is measured with respect to a random execution of P_1 over random inputs where the first-round incoming message, $\pi^1_{1,P_2\to P_1}$, is fixed to its value in the "main" execution. (The latter is a random variable which is induced by the inputs, y and R_2 of P_2.) For every fixing of $\pi^1_{1,\overline{P}\to P}$ the following holds: If the query is light, then the probability of the event $(\star)$ is at most $1/(q_1\rho)$ (by definition). On the other hand, the probability that the response to a heavy query does not appear in the list L^{P_2} that was sampled by P_2 (and was sent in Π^2) is at most $(1-1/(q_1\rho))^{q_1\rho\lambda} \le e^{-\lambda}$. Overall, for any given query the event $(\star)$ happens with probability at most $1/(q_1\rho) + e^{-\lambda}$. Applying union-bound over all q_1 queries, we get that G_1 happens except with probability $q_1(p + 1/(q_1\rho) + e^{-\lambda})$. Overall, we get that

$$\Pr[G] \ge \Pr[G_1 \wedge G_2 \wedge I] \ge 1 - \big(p + 2q_1(p + 1/(q_1\rho) + e^{-\lambda})\big) \ge 1 - O(1/\rho).$$

We now argue security. Notice that the only difference in the transcripts between the protocols Π^1 and Π^2 is in this additional lists (L^{P_1}, L^{P_2}). We show that there exists an efficient, $\mathsf{poly}(\lambda)$-query, algorithm $\mathcal{A}$ that, given the transcript of Π^1 and an oracle access to $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$, can generate a pair of lists that are identical to the lists (L^{P_1}, L^{P_2}) sent in the protocol except with probability $O(1/\rho(\lambda))$ (over the choice of the oracles). Formally, for any fixed inputs x, y and randomness R_1, R_2 of the parties in the original protocol Π^1, with probability of $1 - O(1/\rho(\lambda))$ over the choice of the oracles $\mathcal{O} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$, it holds that

$$\mathcal{A}(\mathbb{T}^1) = (L^{P_1}, L^{P_2}),$$

where $\mathbb{T}^1$ is the transcript of Π^1 and (L^{P_1}, L^{P_2}) are the lists sent in Π^2 where in both executions x, y and R_1 and R_2 are being used and the randomness is induced only by the choice of $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$.

Before describing $\mathcal{A}$, we claim that its existence implies security over randomly chosen inputs. For concreteness, we focus on security against P_1. (The argument for P_2 is similar). Let $\mathsf{view}^{\Pi^1}_{P_1}(x, y)$ (resp., $\mathsf{view}^{\Pi^2}_{P_1}(x, y)$) denote the view of P_1 when the protocol Π^1 (resp., Π^2) is invoked over inputs (x, y) and oracles $\mathcal{O}$. Consider the mapping $\mathcal{A}'$ that, given v^1 (presumably $\mathsf{view}^{\Pi^1}_{P_1}(x, y)$), recovers the lists (L^{P_1}, L^{P_2}) by applying $\mathcal{A}$ on the transcript part of v^1, and outputs v^1 extended with the lists (L^{P_1}, L^{P_2}). Let x and y_0 be random inputs in X_λ and Y_λ, and let y_1 be a random sibling of y_0, i.e., y_1 is uniformly distributed among all strings $y \in Y_\lambda$ for which $f(x, y) = f(x, y_0)$. Note that the marginal distribution of (x, y_1) is also uniform over $X_\lambda \times Y_\lambda$. Next observe that for every $b \in \{0, 1\}$, the random variables

$$(x, y_0, y_1, \mathcal{A}'(\mathsf{view}^{\Pi^1}_{P_1}(x, y_b)), \mathcal{O}) \qquad \text{and} \qquad (x, y_0, y_1, \mathsf{view}^{\Pi^2}_{P_1}(x, y_b), \mathcal{O}) \tag{2.3.1}$$

are identically distributed whenever (1) the output of $\mathcal{A}$ is the same as the lists (L^{P_1}, L^{P_2}) sent by the parties on the real transcript; and (2) the transcript $\mathbb{T}^1$ together with the oracles satisfy the event G (defined above). By union bound, both events happen except with probability $O(1/\rho(\lambda))$ over a random choice of the inputs and the oracles. Hence, for every $b \in \{0, 1\}$,

the random variables in (2.3.1) are $O(1/\rho(\lambda))$-close in statistical distance. That is, even unbounded adversaries that can make unbounded number of queries to the oracles $\mathcal{O}$ cannot distinguish between $(x, y_0, y_1, \mathcal{A}'(\mathsf{view}^{\Pi^1}_{P_1}(x, y_b)))$ and $(x, y_0, y_1, \mathsf{view}^{\Pi^2}_{P_1}(x, y_b))$ with advantage better than $O(1/\rho(\lambda))$. Together with the privacy of Π^1 (over worst-case inputs), this implies that $(x, y_0, y_1, \mathsf{view}^{\Pi^2}_{P_1}(x, y_0))$ cannot be distinguished from $(x, y_0, y_1, \mathsf{view}^{\Pi^2}_{P_1}(x, y_1))$ with advantage better than $\varepsilon + O(1/\rho(\lambda))$ by $(S - \mathsf{poly}(\lambda))$-query distinguishers. Indeed, a distinguisher $D^{\mathcal{O}}(x, y_0, y_1, v^2)$ that violates the above gives rise to a distinguisher $D'^{\mathcal{O}}(x, y_0, y_1, v^1) := D^{\mathcal{O}}(x, y_0, y_1, \mathcal{A}'(v^1))$ that violates the security of Π^1.

We move on to describe the algorithm $\mathcal{A}$. Roughly speaking, the algorithm $\mathcal{A}$ generates the list L^P just like in protocol Π^2, except that calls to OT_3 will not be emulated, and instead will be answered directly by querying the oracle OT_3. Specifically, $\mathcal{A}$ extracts from the transcript the first round message, $\pi^1_{1,P\to\overline{P}}$, sent by a party $P \in \{P_1, P_2\}$ in Π^1, and executes $t = q_1\lambda\rho$ independent executions of the other party $\overline{P}$ with uniformly chosen input and randomness. It then executes the second message function of $\overline{P}$ in the protocol Π^1 with respect to the incoming message $\pi^1_{1,P\to\overline{P}}$. During these executions $\mathcal{A}$ delivers OT_3-queries to the OT_3 oracle. For every such valid query made to the OT_3 oracle (i.e., the response is not $\perp$) whose r part has length greater than λ' and for which otm_2 is not a response obtained from OT_2 in this execution, it will add $((\mathsf{otm}_1, \mathsf{otm}_2, b, r), m_b)$ to the list L^P of party P where m_b is the response from OT_3.

We analyze $\mathcal{A}$. Fix the inputs x, y the randomness R_1, R_2 of the parties in the main execution (of Π^1) and the randomness that is being used in the auxiliary executions. Let us assume that the oracles are injective. In this case, the simulation deviates from the real interaction only if the simulator (resp. the real execution) adds an entry $((\mathsf{otm}_1, \mathsf{otm}_2, (b, r)), m_b)$ to the list L^P although this entry is not being added in the real execution (resp. the simulation). This happens only if the corresponding OT_3-query does not evaluate to $\perp$ (by the oracle OT_3) although its otm_2-part is not found in the OT_2-lists of both parties that were generated by the first round functions during the corresponding random execution of Π^1. By Fact 2.3.6, this event happens with probability at most p per OT_3 query. Applying a union bound over all queries (in all emulations of both parties) and on the event that the oracles fail to be injective, we conclude that, except with probability $p + 2ptq_1 < O(1/\rho)$, all the entries in the lists generated by $\mathcal{A}$ are identical to the entries in the lists generated in the protocol. ■

2.3.2.4 Transformation-3: $\Pi^2 \Rightarrow \Pi^*$

We now transform Π^2 into Π^* that computes g in $O(n)$ rounds in the presence of oracles $(\mathsf{OT}_1, \mathsf{OT}_2)$ without making OT_3 queries. We will make use of the following "dependency learner" $\mathcal{E}$ that was defined by Barak and Mahmoody [BM09].

Lemma 2.3.14 ([BM09]) *Let $\mathcal{O}$ be a set of random oracles. Let $\Pi^{\mathcal{O}}$ be an interactive protocol between P_1 and P_2 in which they might use private randomness (but no inputs otherwise)*

and they each ask at most m queries to $\mathcal{O}$. Then, there is a deterministic eavesdropping algorithm $\mathcal{E}$ (whose algorithm might depend on P_1 and P_2) who gets as input a parameter $\varepsilon \in (0,1)$ and the transcript $\mathbb{T}$ of a random execution of the protocol $\Pi^{\mathcal{O}}$ (with the views of P_1 and P_2 being random variables view_{P_1} and view_{P_2}), asks at most $\mathsf{poly}(m/\varepsilon)$ queries to the random oracles $\mathcal{O}$, such that the probability that P_1 and P_2 have an "intersection query" outside of the queries asked by Eve to any oracle $\mathsf{O} \in \mathcal{O}$ is at most ε. That is,

$$\forall \mathsf{O} \in \mathcal{O}, \qquad \Pr[Q^{\mathsf{O}}(\mathsf{view}_{P_1}) \cap Q^{\mathsf{O}}(\mathsf{view}_{P_2}) \not\subseteq Q^{\mathsf{O}}(\mathsf{view}_{\mathcal{E}})] \leq \varepsilon,$$

where $Q^{\mathsf{O}}(\mathsf{view}_P)$ denotes the (random variable) that consists of all the oracle queries that were asked by P.

We give the description of Π^* below.

Transformation $\Pi^2 \Rightarrow \Pi^*$:

- **Parameters:** Let $q_2 = q_2(\rho)$ be the number of oracle queries that each party makes in the first and the second rounds of Π^2 and let $n = n(\lambda)$ be the number of oracle queries made by the decoder. Let $\rho = \rho(\lambda)$ be an arbitrary polynomial and set $p = \frac{1}{(q_2+n^2)\rho}$. Let $\varepsilon' = 1/(2n\rho)$.
- **Preprocessing Phase:** For every $\lambda' \leq \log(1/p)$, each party $P \in \{P_1, P_2\}$ makes oracle queries to $\mathsf{OT}_2^{\lambda'}$ on every point in the domain. Party P creates a list L_2^P with the queries/responses to the OT_2 oracle.
- **Executing Π^2:** Each party $P \in \{P_1, P_2\}$ in Π^* executes the first and the second round messages of $\Pi^2[\rho]$ while updating the list L_2^P with the queries/responses to the OT_2 oracle. Let $\mathbb{T}$ denote the resulting transcript for the first two rounds.
- **Calling the dependency learner $\mathcal{E}$:** P_1 invokes the dependency learner $\mathcal{E}$ for the protocol Π^2 (which is promised by Lemma 2.3.14) over the transcript $\mathbb{T}$ with proximity parameter ε'. Let L_2^E be the list of OT_2 queries made by $\mathcal{E}$ to the OT_2 oracle.
- **Executing the decoder Dec^2:** Next, P_1 invokes the Π^2-decoder Dec^2 over the transcript $\mathbb{T}$. (Recall that the decoder is deterministic.) For every $\mathsf{OT}_1, \mathsf{OT}_2$ queries made by Dec^2, P_1 maintains a list of the queries/responses obtained. For every OT_3 query on $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ made by Dec^2, P_1 gives the response as follows:
  1. It makes an OT_1 oracle call on (b, r) and checks if the response is otm_1. If not, it outputs $\perp$ as the response to the OT_3 query.
  2. If $|r| \leq \log(1/p)$, this query can be answered by an exhaustive search on the list L_2^P as it contains all the responses to every point in the domain of OT_2.
  3. Else, P_1 checks if there is an entry $((\mathsf{otm}_1, m_0, m_1, s), \mathsf{otm}_2)$ in the list of queries/responses to the OT_2 oracle made by Dec^2. If such an entry is found, it outputs m_b.

4. Else, P_1 checks if there is an entry $((\mathsf{otm}_1, m_0, m_1, s), \mathsf{otm}_2)$ in the $L_2^{P_1}$ or L_2^E list. If yes, it gives m_b as the response.
5. Else, P_1 sends $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ to P_2. The other party P_2 sends m_b if an $(\mathsf{otm}_1, (m_0, m_1, s), \mathsf{otm}_2)$ entry appears in its $L_2^{P_2}$ list, and otherwise sends $\perp$. P_1 responds to decoder's query with this message.

- At the end of the execution, the decoder Dec^2 outputs a value z (supposedly $g(x, y)$) and P_1 sends it as the last round message. Both parties output z.

We show that the protocol Π^* is a $(\delta - O(1/\rho(\lambda)), \varepsilon - O(1/\rho(\lambda)), S - \mathsf{poly}(\rho, \lambda))$-secure protocol that computes g over random inputs, thus proving Lemma 2.3.8.

The following claim shows that the emulation of the decoder is typically consistent with OT_3, and therefore establishes the correctness of Π^*.

Claim 2.3.15 (Correctness) *For every fixing of the inputs and local private tapes, except with probability $O(1/n\rho(\lambda))$ over the choice of the oracles, all the OT_3 queries of Dec^2 are answered consistently with the answers of the real OT_3 oracle. Consequently, for uniformly chosen inputs (x, y) the output of both parties Dec^* equals to $g(x, y)$ with probability at least $\delta - O(1/\rho(\lambda))$.*

Proof We begin by proving the first part. As before, if $\mathsf{OT}_1(b, r) \neq \mathsf{otm}_1$ or if $|r| \leq \log(1/p)$, then the emulation is perfect. We now show that, except with probability $O(1/n\rho(\lambda))$, the emulated answers to all the queries for which $r > \log(1/p)$ are consistent with the answers of the real oracle. Let us first assume that the oracles $\mathsf{OT}_1, \mathsf{OT}_2$ are injective which, by Fact 2.3.5, happens except with probability p. Observe that when the output of the emulation is not $\perp$, it is consistent with the $\mathsf{OT}_1, \mathsf{OT}_2$ queries and, therefore, by injectivity, must be also consistent with OT_3. Hence, the emulation may disagree with the real oracle only when the emulated output is $\perp$. Let us therefore consider a query $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ for which the emulation returns $\perp$. Recall that this happens only if otm_2 does not appear in any of the L_2 lists of $P_1, P_2, \mathcal{E}$ and the decoder. Therefore, by Fact 2.3.6, the response from the actual OT_3 oracle will also be $\perp$ except with probability p over the choice of OT_2. Overall, by a union bound, the emulation agrees with the oracle on all queries except with probability $p + np \leq O(1/n\rho(\lambda))$.

To prove the "consequently" part, observe that the only difference between Π^2 and Π^* is how the OT_3 queries of Dec^2 are answered. Hence, conditioned on the event that the emulation is perfect, the output of Π^* on (x, y) is equal to the output of Π^2 over (x, y), and so the claim follows from the average-case correctness of Π^2 (as proved in Claim 2.3.13). ■

We now show the privacy.

Claim 2.3.16 (Privacy) *For every non-uniform distinguishers D_1, D_2 making at most $S - \mathsf{poly}(\lambda)$ queries to the oracles $\mathcal{O}' = (\mathsf{OT}_1, \mathsf{OT}_2)$ it holds that:*

$$|\Pr[D_1^{\mathcal{O}'}((y, y'), \mathsf{view}_{P_1}(x, y)) = 1] - \Pr[D_1^{\mathcal{O}'}((y, y'), \mathsf{view}_{P_1}(x, y')) = 1]| \leq \varepsilon + O(1/\rho(\lambda))$$

$$|\Pr[D_2^{\mathcal{O}'}((x,x'),\mathsf{view}_{P_2}(x,y))=1]-\Pr[D_2^{\mathcal{O}'}((x,x'),\mathsf{view}_{P_2}(x',y))=1]|\leq\varepsilon+O(1/\rho(\lambda)),$$

where $(x,y)\leftarrow X_\lambda\times Y_\lambda$, $y'\in Y_\lambda$ *is chosen uniformly from the set* $\{y':g(x,y')=g(x,y)\}$, $x'\in X_\lambda$ *is chosen uniformly from the set* $\{x':g(x',y)=g(x,y)\}$, *and* $\mathsf{view}_{P_i}(x,y)$ *denotes the view of* P_i *in the execution of* Π^* *on inputs* (x,y).

2.3.2.5 Proof of Claim 2.3.16

For $i\in[n]$, we define a hybrid protocol Π_i^* which differs from Π^* only in the way that the OT_3-queries of the decoder are being answered. Specifically, the first i queries are answered exactly as in Π^*, and in the last $n-i$ queries we modify Step 5 and instead of sending the query $(\mathsf{otm}_1,\mathsf{otm}_2,(b,r))$ to P_2, we send it to the OT_3 oracle (and deliver the response to the decoder).

Notice that Π_n^* is identical to Π^*. We also show that Π_0^* inherits its security from Π^2. In the following, let $\mathsf{view}_P^i(x,y)$ be the view of P in an execution of $\Pi_i^*(x,y)$ and let $\mathsf{view}_P^{\Pi^2}(x,y)$ be the view of P in an execution of $\Pi^2(x,y)$.

Claim 2.3.17 *There exist efficient transformations* $\mathcal{A}_1,\mathcal{A}_2$ *that make no calls to the* $\mathcal{O}=(\mathsf{OT}_1,\mathsf{OT}_2,\mathsf{OT}_3)$*-oracles such that for* any x,y, *for any choice of the random tapes for the parties, and for* $i\in\{1,2\}$ *the random variables*

$$(\mathcal{A}_i(\mathsf{view}_{P_i}^{\Pi^2}(x,y)),\mathcal{O})\qquad\text{and}\qquad(\mathsf{view}_{P_i}^{0}(x,y),\mathcal{O})$$

are $O(1/n\rho(\lambda))$*-close (over the choice of the oracles).*

Note that the above means that indistinguishability holds even with respect to an unbounded distinguisher that makes unbounded number of queries to the oracles $\mathcal{O}$.

Proof Let us fix the inputs x,y and the random tapes of the parties, and conditioned on the event I that the oracle OT_2 is injective on inputs whose s part has length greater than $\log(1/p)$. By Fact 2.3.5, this event happens except with probability $p<O(1/n\rho(\lambda))$.

The protocol $\Pi_0^*(x,y)$ is identical to $\Pi^2(x,y)$ with two minor modifications. Starting with P_1, the only difference is that in Π^2 when P_1 runs the decoder Dec^2 all the OT_3 queries are answered by the OT_3 oracle, whereas in Π_0^* the OT_3 queries that reach to Steps 1–4 are answered locally. (This means that, syntactically, the view of P_1 in Π^2 contains the answers to all these queries whereas the view in Π_0^* does not contain the corresponding responses.) Observe that, under I, the answers obtained in both cases are the same. Therefore the transformation $\mathcal{A}_1$ that takes $\mathsf{view}_{P_1}^{\Pi^2}(x,y)$ and removes the responses to the OT_3 queries that reach Steps 1–4, satisfies the claim with statistical deviation of p.

We move on to P_2. In Π_0^*, party P_2 gets the output z from P_1 at the end, whereas in Π^2, party P_2 invokes the decoder Dec^2 locally on the transcript. This means that, syntactically, the view of P_2 in Π^2 contains the answers to all the decoder's queries whereas the view in Π_0^* does not contain the corresponding responses but contains the message z. Consider the transformation $\mathcal{A}_2$ that takes $\mathsf{view}_{P_2}^{\Pi^2}(x,y)$, computes the output z' that is induced by

the view, appends z to $\mathsf{view}^{\Pi^2}_{P_1}(x,y)$ and removes all the responses to the queries made by Dec^2. Again, whenever I happens, the simulated outcome z' is equal to the actual message z sent in Π^2. Therefore, by Fact 2.3.5, $\mathcal{A}_2$ satisfies the claim with statistical deviation of $p < O(1/n\rho(\lambda))$. ■

We show that privacy is approximately preserved when moving between neighboring hybrids.

Claim 2.3.18 *For every $i \in [n]$ and every $\varepsilon' > 0$, there exist two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ that make at most $\mathsf{poly}(\lambda, 1/\delta', 1/\varepsilon')$ queries to $\mathcal{O} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ such that, for uniformly chosen $(x,y) \leftarrow X_\lambda \times Y_\lambda$, the random variables*

$$(y, \mathcal{A}_1(\mathsf{view}^{i-1}_{P_1}(x,y)), \mathcal{O}) \qquad \textit{and} \qquad (y, \mathsf{view}^{i}_{P_1}(x,y), \mathcal{O})$$

are $O(1/n\rho(\lambda))$-close (in statistical distance) and

$$(x, \mathcal{A}_2(\mathsf{view}^{i-1}_{P_2}(x,y)), \mathcal{O}) \qquad \textit{and} \qquad (x, \mathsf{view}^{i}_{P_2}(x,y), \mathcal{O})$$

are $O(1/n\rho(\lambda))$-close.

Proof We start with the description of $\mathcal{A}_1$. On input $\mathsf{view}^{i-1}_{P_1}$, the algorithm $\mathcal{A}_1$ emulates the execution of P_1 in Π^*_i up to the generation of the i-th OT_3 query $(\mathsf{otm}_1, \mathsf{otm}_2, (b,r))$ of the decoder. At this point, it checks whether one of the conditions (1–4) holds. If this is the case, then it outputs $\mathsf{view}^{i-1}_{P_1}$ as is. Otherwise, if the emulation reaches Step 5, the algorithm does the following:

- It queries the OT_3 oracle on $(\mathsf{otm}_1, \mathsf{otm}_2, (b,r))$ and gets a response a.
- It adds $(\mathsf{otm}_1, \mathsf{otm}_2, (b,r))$ as the message from P_1 to P_2 to the transcript $\mathbb{T}_{i-1}$ (extracted from $\mathsf{view}^{i-1}_{P_1}$) and it also adds the response obtained from OT_3 oracle as the incoming message sent from P_2 to P_1 in $\mathbb{T}_{i-1}$. Let $\mathbb{T}^*_{i-1}$ be the augmented transcript.

Analysis of $\mathcal{A}_1$. Fact 2.3.5 implies that OT_1 and OT_2 are injective on inputs of length $\geq \log(1/p)$ with probability at least $1-p$. We condition on this event and prove that the tuple $(y, \mathcal{A}_1(\mathsf{view}^{i-1}_{P_1}), \mathcal{O})$ is p-close to the tuple $(y, \mathsf{view}^{i}_{P_1}, \mathcal{O})$. Notice that the only difference between $\mathcal{A}_1(\mathsf{view}^{i-1}_{P_1})$ and $\mathsf{view}^{i}_{P_1}$ is in answering the i-th oracle query of the decoder and the corresponding messages exchanged. We show that the response of the i-th query to the decoder and the messages exchanged while answering this query are identical in the two distributions except with probability p. Let the i-th oracle query of the decoder be $(\mathsf{otm}_1, \mathsf{otm}_2, (b,r))$. Observe that the only case where $\mathcal{A}_1(\mathsf{view}^{i-1}_{P_1})$ differs from $\mathsf{view}^{i}_{P_1}$ is when otm_2 is in the image of OT_2 oracle, but it is not in the list of queries made by either parties, $\mathcal{E}$ or Dec^2. However, by Fact 2.3.6, the probability that this event happens (over the choice of the oracles) is at most p. Overall, the deviation is $2p \leq O(1/n\rho(\lambda))$.

Description of $\mathcal{A}_2$. We now give the description of $\mathcal{A}_2$. Given an input $\mathsf{view}_{P_2}^{i-1}$, the algorithm $\mathcal{A}_2$ parses the view into the input y and random tape r, the transcript $\mathbb{T}$ corresponding to the first two rounds of Π_{i-1}^*, the transcript $\mathbb{T}_{i-1}$ of the remaining rounds of Π_{i-1}^*, and the last message z (presumably the output) sent by P_1. The algorithm $\mathcal{A}_2$ outputs $(y, r, \mathbb{T}_i, z)$ where $\mathbb{T}_i$ is generated as follows.

1. $\mathcal{A}_2$ runs $\mathcal{E}$ on $\mathbb{T}$ and let L_2^E be the list of OT_2 queries made by $\mathcal{E}$. In addition, $\mathcal{A}_2$ starts running (the deterministic) Dec^2 on the transcript $\mathbb{T}$. The first $i-1$ queries that Dec^2 makes to the oracle OT_3 are being answered directly by using the OT_3 oracle. Given the i-th OT_3 query, $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$, of the decoder Dec^2, the algorithm $\mathcal{A}_2$ proceeds as follows.

2. If one of the following conditions hold, set $\mathbb{T}_i = \mathbb{T}_{i-1}$:

   a) The evaluation of the OT_1 oracle on (b, r) is not equal to otm_1.

   b) $|r| \leq \log(1/p)$.

   c) There is an entry $((\mathsf{otm}_1, m_0, m_1, s), \mathsf{otm}_2)$ in the list of queries/responses to the OT_2 oracle made by Dec^2.

   d) There is an entry $((\mathsf{otm}_1, m_0, m_1, s), \mathsf{otm}_2)$ in the L_2^E list.

3. Otherwise, if there is an entry $((\mathsf{otm}_1, m_0, m_1, s), \mathsf{otm}_2)$ in the $L_2^{P_2}$ list, then $\mathcal{A}_2$ appends $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ as message from P_1 to P_2 and m_b as the response from P_2 to P_1 to the transcript $\mathbb{T}_{i-1}$ to obtain $\mathbb{T}_i$.

4. Otherwise (if such an entry is not found in either lists), $\mathcal{A}_2$ queries the OT_3 oracle on $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ and obtains the response. If the response was $\perp$, it adds $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$ as message from P_1 to P_2 and $\perp$ as the response from P_2 to P_1 to the transcript $\mathbb{T}_{i-1}$ to obtain $\mathbb{T}_{i-1}^*$. Otherwise, it sets $\mathbb{T}_i = \mathbb{T}_{i-1}$.

Analysis of $\mathcal{A}_2$. Consider the random experiment in which the inputs x and y are chosen at random, as well as the random tapes of both parties and the oracles $\mathcal{O}$. Let us condition on the following events: (1) The oracles OT_1 and OT_2 are injective on inputs of length $\geq \log(1/p)$; and (2) The emulation of the decoder up to step i by $\mathcal{A}_2$ is consistent with its emulation by P_1 in Π_{i-1}^*. By Fact 2.3.5 and Claim 2.3.15 both events happen except with probability $p + O(1/n\rho(\lambda)) \leq O(1/n\rho(\lambda))$. Let us condition on these events and prove that the tuple $(x, \mathcal{A}_2(\mathsf{view}_{P_1}^{i-1}), \mathcal{O})$ is $(\varepsilon' + p)$-close to the tuple $(x, \mathsf{view}_{P_1}^{i}, \mathcal{O})$.

Observe that, under these events, the i-th OT_3 query of the decoder, $(\mathsf{otm}_1, \mathsf{otm}_2, (b, r))$, is identical in both the simulation (performed by $\mathcal{A}_2$) and the real protocol Π_{i-1}^*. Hence, deviation may happen only if either (A) the simulation appends the query to the transcript but the real execution does not; or (B) the real-execution appends the query to the transcript but the simulation does not.

We start with Case (A). The simulation appends the query only in Steps 3 and 4. In the latter case (Step 4), the real execution also appends the query. Hence, Case (A) may

happen only in Step 3, which means that the query is an intersecting query (it appears both in $L_2^{P_1}$ and $L_2^{P_2}$) but it was not detected by $\mathcal{E}$ (does not appear in L_2^E). By Lemma 2.3.14, this event happens with probability at most ε'.

Finally, observe that Case (B) may happen only if the simulation reaches to Step 4 and the query is not evaluated to $\perp$. This means that otm_2 is in the image of OT_2 oracle, but it is not in the list of queries made by any of the parties, which, by Fact 2.3.6, happens with probability at most p (over the choice of the oracles). ■

We can now complete the proof of Claim 2.3.16. First, by iteratively composing Claim 2.3.18 and applying Claim 2.3.17, there exists an algorithm $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) that makes at most $n\mathsf{poly}(\lambda) = \mathsf{poly}(\lambda)$ queries to the oracles $\mathcal{O}$ such that for every $b \in \{1,2\}$, the random variables

$$(x, y, \mathcal{A}_b(\mathsf{view}_{P_b}^{\Pi^2}(x,y)), \mathcal{O}) \qquad \text{and} \qquad (x, y, \mathsf{view}_{P_b}(x,y), \mathcal{O})$$

are $O(1/\rho(\lambda))$-close where (x, y) are uniformly distributed and view stands for the view under Π^*. (The above notation is somewhat redundant since for $b = 1$ the random variable x appears as part of P_1's view and for $b = 2$ the random variable y appears as part of P_2's view.)

Next, consider a distinguisher D that attacks the privacy of P_1 in Π^* with respect to random inputs. (The case of P_2 is symmetric.) That is, D makes at most S' queries to $\mathcal{O}'$ and

$$|\Pr[D^{\mathcal{O}'}((y,y'), \mathsf{view}_{P_1}(x,y)) = 1] - \Pr[D^{\mathcal{O}'}((y,y'), \mathsf{view}_{P_1}(x,y')) = 1]| > \Delta,$$

where $(x, y) \leftarrow X_\lambda \times Y_\lambda$, and $y' \in Y_\lambda$ is chosen uniformly from the set $\{y' : g(x, y') = g(x, y)\}$. It follows that the adversary $D'^{\mathsf{OT}_1,\mathsf{OT}_2,\mathsf{OT}_3}(y, y', v)$ that computes $v' = \mathcal{A}_1^{\mathsf{OT}_1,\mathsf{OT}_2,\mathsf{OT}_3}(v)$ and then outputs $D^{\mathsf{OT}_1,\mathsf{OT}_2}(y, y', v')$ satisfies

$$|\Pr[D'^{\mathcal{O}}((y,y'), \mathsf{view}_{P_1}^{\Pi^2}(x,y)) = 1] - \Pr[D'^{\mathcal{O}}((y,y'), \mathsf{view}_{P_1}^{\Pi^2}(x,y')) = 1]| > \Delta - O(1/\rho(\lambda)).$$

(Here we use the fact that the marginal distribution of (x, y') is also uniform over $X_\lambda \times Y_\lambda$.) Since D' makes at most $S'+\mathsf{poly}(\lambda)$ queries to $\mathcal{O}$, Claim 2.3.13 implies that for some constant c, whenever $S' < S-\lambda^c$ the distinguishing advantage Δ is at most $\varepsilon+O(1/\rho(\lambda))$. Claim 2.3.16 follows.

2.4 Separating Non-Compact HSS from 2-Round OT

In this section, we present a simple corollary of our main separation to a natural variant of *homomoprhic secret sharing* (HSS). Concretely, we show a black-box separation between a weak 2-party flavor of "non-compact HSS" and 2-round oblivious transfer. Non-compact (2-party) HSS is similar to the notion of 2-input HSS from [Boy+18] in that it allows a local computation of a function $g(x, y)$ on independently generated shares of x and y. However, instead of the usual requirement that the output shares be *additive* or *compact*, here we require that they reveal no additional information except the output.

Definition 2.4.1 (Non-compact HSS) *A 2-party* non-compact homomorphic secret sharing *(non-compact HSS) for a function $g : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a triple of PPT algorithms* $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ *with the following syntax.*

- $\mathsf{Share}(x)$ *takes an input $x \in \{0,1\}^\lambda$ and outputs two shares (x_1, x_2).*
- $\mathsf{Eval}(j, (x_j, y_j))$ *takes in $j \in \{1,2\}$ and the j-th shares of the inputs and outputs z_j (an output share).*
- $\mathsf{Dec}(z_1, z_2)$ *is a deterministic algorithm that takes output shares z_1, z_2 and outputs a string z.*

We say that $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ *is a (δ, ε)-non-compact HSS for computing g if it satisfies the following two properties:*

- ***Correctness.*** *For any $x, y \in \{0,1\}^\lambda$, we require*

$$\Pr[\mathsf{Dec}(z_1, z_2) = f(x, y)] \geq \delta(\lambda)$$

  where $(x_1, x_2) \leftarrow \mathsf{Share}(x)$, $(y_1, y_2) \leftarrow \mathsf{Share}(y)$ and $z_j \leftarrow \mathsf{Eval}(j, (x_j, y_j))$ for $j \in \{1,2\}$.

- ***Security.*** *There exist PPT $\mathsf{Sim}_1, \mathsf{Sim}_2$, and Sim such that for every input $x, y \in \{0,1\}^\lambda$ and every efficient nonuniform distinguishers D_1, D_2, and D the following holds:*

$$|\Pr[D_1(x, (r_1, s_1), y_1, z_2)] - \Pr[D_1(\mathsf{Sim}_1(x, f(x, y)))]| \leq \varepsilon(\lambda)$$

$$|\Pr[D_2((y, (r_2, s_2), x_2, z_1))] - \Pr[D_2(\mathsf{Sim}_1(y, f(x, y)))]| \leq \varepsilon(\lambda)$$

  and

$$|\Pr[D(z_1, z_2)] - \Pr[D(\mathsf{Sim}_1(f(x, y)))]| \leq \varepsilon(\lambda)$$

  where $(x_1, x_2) \leftarrow \mathsf{Share}(x; r_1)$, $(y_1, y_2) \leftarrow \mathsf{Share}(y; r_2)$ and $z_j \leftarrow \mathsf{Eval}(j, (x_j, y_j); s_j)$.

When the error parameters ε, δ are omitted, they are understood to be negligible.

Previous definitions of HSS [BGI16; Boy+18] require that each *input* share hide the input, and make no security requirements involving the output shares. However, we note that the default syntactic requirement for HSS schemes as defined in [BGI16; Boy+18] is that the output decoder computes *addition* over a finite Abelian group. This type of additive HSS implies our notion of non-compact HSS by masking the two output shares with a random group element. Thus, the following separation result applies also to additive HSS. Finally, note that the above definition does not explicitly require that each input share hide the input. However, for nontrivial functions g this is implied by the above requirements, and in any case not requiring this makes the negative result stronger.

Towards separating non-compact HSS from 2-round OT, we use the following simple transformation of HSS to a 2-round protocol with publicly decodable transcript.

Claim 2.4.2 *For any $\delta, \varepsilon > 0$, (δ, ε)-non-compact HSS for computing g in the can be used as a black box to construct a protocol (Π, Dec) for computing g in 2 rounds with (δ, ε)-publicly decodable transcript.*

Proof We start with the description of Π. In the first round, the party $P \in \{P_1, P_2\}$ runs Share on its input and sends the other party's share. The parties then run Eval on the shares to obtain z_1 and z_2 respectively. In the second round, the parties exchange z_1 and z_2. We define Dec to extract (z_1, z_2) and run the decoder for HSS to learn $g(x, y)$. The security properties directly follow from the properties of non-compact HSS. ■

Using the black-box separation of Corollary 2.3.4, we get the following corollary.

Corollary 2.4.3 *There is no black-box construction of non-compact 2-party HSS for $g(x_1, x_2) = x_1 \wedge x_2$ from 2-round OT.*

In contrast, a non-black-box construction of non-compact 2-party HSS from 2-round OT easily follows from the positive result for the 2-round client-server model from [GIS18] via a simple syntactic translation. Indeed, consider such a protocol for a function $g(x, y)$ with 2 input clients, two servers, and a single output client. The interaction in such a protocol consists of a message from each input client to each server, followed by a message from each server to the output client. Assume without loss of generality that the two input clients run the same algorithm. Then $\mathsf{Share}(x)$ outputs the pair of messages sent by a client on input x, $\mathsf{Eval}(j, (x_1, x_2))$ the message sent by server j upon receiving messages (x_j, y_j) from the client, and $\mathsf{Dec}(z_1, z_2)$ returns the output of the output client upon receiving messages z_1, z_2.

Chapter 3

Non-Black-Box Construction

In this chapter, we give a non-black-box construction of two-round secure multiparty computation from two-round oblivious transfer. In Section 3.1, we give a brief overview of the universal composability framework. In Section 3.2, we define the main building blocks used in our construction. In Section 3.3, we define conforming protocols which is an intermediate primitive that is used in our construction of two-round protocols. In Section 3.4, we give a construction of two-round MPC protocol in the plain model against semi-honest adversaries and in Section 3.5, we give a two-round MPC protocol in the Common Random/Reference String model with security against malicious adversaries.

3.1 Universal Composability Framework

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

3.1.1 The basic model of execution

Following [GMR88; Gol04], a protocol is represented as an interactive Turing machine (ITM), which represents the program to be run within each participant. Specifically, an ITM has three tapes that can be written to by other ITMs: the input and subroutine output tapes model the inputs from and the outputs to other programs running within the same "entity" (say, the same physical computer), and the incoming communication tapes and outgoing communication tapes model messages received from and to be sent to the network. It also has an identity tape that cannot be written to by the ITM itself. The identity tape contains the program of the ITM (in some standard encoding) plus additional identifying information specified below. Adversarial entities are also modeled as ITMs.

We distinguish between ITMs (which represent static objects, or programs) and *instances of ITMs*, or ITIs, that represent interacting processes in a running system. Specifically, an

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

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

With one exception (discussed within) we assume that all ITMs are probabilistic polynomial time (PPT). An ITM is PPT if there exists a constant $c > 0$ such that, at any point during its run, the overall number of steps taken by M is at most n^c, where n is the overall number of bits written on the *input tape* of M in this run. (In fact, in order to guarantee that the overall protocol execution process is bounded by a polynomial, we define n as the total number of bits written to the input tape of M, *minus the overall number of bits written by M to input tapes of other ITMs.*; see [Can01].)

3.1.2 Security of protocols

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

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

The environment is given an external input z and is the first to be activated. In its first activation, the environment invokes the adversary $\mathcal{A}$, providing it with some arbitrary input. In the context of UC security, the environment can from now on invoke (namely, provide

input to) only ITMs that consist of a single instance of protocol Π. That is, all the ITMs invoked by the environment must have the same SID and the code of Π.

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

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

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

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

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

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

Securely realizing an ideal functionality. We say that a protocol Π *emulates* protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running Π, or it is interacting with S and parties running ϕ. This means that, from the

point of view of the environment, running protocol Π is 'just as good' as interacting with ϕ. We say that Π *securely realizes* an ideal functionality $\mathcal{F}$ if it emulates the ideal protocol $\Pi(\mathcal{F})$. More precise definitions follow. A distribution ensemble is called *binary* if it consists of distributions over $\{0,1\}$.

Definition 3.1.1 *Let Π and ϕ be protocols. We say that Π UC-emulates ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.*

Definition 3.1.2 *Let $\mathcal{F}$ be an ideal functionality and let Π be a protocol. We say that Π UC-realizes $\mathcal{F}$ if Π UC-emulates the ideal process $\Pi(\mathcal{F})$.*

3.1.3 Hybrid protocols

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

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

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

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

The universal composition operation. We define the universal composition operation and state the universal composition theorem. Let ρ be an $\mathcal{F}$-hybrid protocol, and let Π be a protocol that securely realizes $\mathcal{F}$. The composed protocol ρ^{Π} is constructed by modifying the code of each ITM in ρ so that the first message sent to each copy of $\mathcal{F}$ is replaced with

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

The universal composition theorem. Let $\mathcal{F}$ be an ideal functionality. In its general form, the composition theorem basically says that if Π is a protocol that UC-realizes $\mathcal{F}$ then, for any $\mathcal{F}$-hybrid protocol ρ, we have that an execution of the composed protocol ρ^{Π} "emulates" an execution of protocol ρ. That is, for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and protocol ρ^{Π} or with $\mathcal{S}$ and protocol ρ, in a UC interaction. As a corollary, we get that if protocol ρ UC-realizes $\mathcal{F}$, then so does protocol ρ^{Π}. [1]

Theorem 3.1.3 (Universal Composition [Can01].) *Let $\mathcal{F}$ be an ideal functionality. Let ρ be a $\mathcal{F}$-hybrid protocol, and let Π be a protocol that UC-realizes $\mathcal{F}$. Then protocol ρ^{Π} UC-emulates ρ.*

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

3.1.4 The Common Reference/Random String Functionality

In the common reference string (CRS) model [CF01; Can+02], all parties in the system obtain from a trusted party a reference string, which is sampled according to a pre-specified distribution D. The reference string is referred to as the *CRS*. In the UC framework, this is modeled by an ideal functionality $\mathcal{F}_{CRS}^{D}$ that samples a string ρ from a pre-specified distribution D and sets ρ as the CRS. $\mathcal{F}_{CRS}^{D}$ is described in Figure 3.1.

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

3.1.5 General Functionality

We consider the general-UC functionality $\mathcal{F}$, which securely evaluates any polynomial-time (possibly randomize) function $f : (\{0,1\}^{\ell_{in}})^n \to (\{0,1\}^{\ell_{out}})^n$. The functionality $\mathcal{F}_f$ is pa-

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

Functionality $\mathcal{F}_{\mathbf{CRS}}^{\mathbf{D}}$

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

1. Upon activation with session id sid proceed as follows. Sample $\rho = D(r)$, where r denotes uniform random coins, and send $(\texttt{crs}, sid, \rho)$ to the adversary.
2. On receiving $(\texttt{crs}, sid)$ from the adversary, send $(\texttt{crs}, sid, \rho)$ to every uncorrupted party.

Figure 3.1: The Common Reference String Functionality.

rameterized with a function f and is described in Figure 3.2. In this paper we will only be concerned with the *static* corruption model.

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

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

1. Each party P_i (and $\mathcal{S}$ on behalf of P_i if P_i is corrupted) sends $(\mathsf{input}, \mathsf{sid}, \mathcal{P}, P_i, x_i)$ to the functionality.
2. Upon receiving the inputs from all parties, evaluate $(y_1, \ldots y_n) \leftarrow f(x_1, \ldots, x_n)$. For every P_i that is corrupted, send adversary $\mathcal{S}$ the message $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, y_i)$.
3. On receiving $(\mathsf{generateOutput}, \mathsf{sid}, \mathcal{P})$ from $\mathcal{S}$, the ideal functionality outputs $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, y_i)$ to P_i for every uncorrupted P_i. If $\mathcal{S}$ sends $(\mathsf{abort}, \mathsf{sid}, \mathcal{P})$, the ideal functionality outputs $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, \perp)$ to P_i for every uncorrupted P_i. (And ignores the message if inputs from all parties in $\mathcal{P}$ have not been received.)

Figure 3.2: General Functionality.

3.2 Building Blocks

In this section, we will describe the main building blocks used in our construction of multi-party computation.

3.2.1 Garbled Circuits

Below we recall the definition of garbling scheme for circuits [Yao86] (see Applebaum et al. [AIK04; AIK05], 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 $(\mathsf{Garb}, \mathsf{Eval})$. Garb is the circuit garbling procedure and Eval is the corresponding evaluation procedure. More formally:

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

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

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

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

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

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

  where $(\widetilde{C}, \{\mathsf{lab}_{w,b}\}_{w\in\mathsf{inp},b\in\{0,1\}}) \leftarrow \mathsf{Garb}\left(1^\lambda, C\right)$ and $\approx_c$ denotes that the two distributions are computationally indistinguishable.
- **Authenticity of Input labels.** For any PPT adversary $\mathcal{A}$, the probability that the following game outputs 1 is negligible.

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

Remark 3.2.1 *We can add authenticity of input labels property generically to any garbled circuit construction by digitally signing every input label and including the verification key as part of the garbled circuit $\widetilde{C}$.*

3.2.2 Oblivious Transfer

In this thesis, we consider a 2-round, 1-out-of-2 *oblivious transfer* protocol (OT), similar to [BM89; NP01; AIR01; HK12] where one party, the *sender*, has input composed of two strings (s_0, s_1) and the input of the second party, the *receiver*, is a bit β. The receiver should learn s_β and nothing regarding $s_{1-\beta}$ while the sender should gain no information about β.

Semi-Honest Two-Round Oblivious Transfer. A two-round semi-honest OT protocol $\langle S, R\rangle$ is defined by three probabilistic algorithms $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ as follows. The receiver runs the algorithm OT_1 which takes the security parameter 1^λ, and the receiver's input $\beta \in \{0,1\}$ as input and outputs ots_1 and ω.[2] The receiver then sends ots_1 to the sender, who obtains ots_2 by evaluating $\mathsf{OT}_2(\mathsf{ots}_1, (s_0, s_1))$, where $s_0, s_1 \in \{0,1\}^\lambda$ are the sender's input messages. The sender then sends ots_2 to the receiver who obtains s_β by evaluating $\mathsf{OT}_3(\mathsf{ots}_2, (\beta, \omega))$.

- **Correctness.** For every choice bit $\beta \in \{0,1\}$ of the receiver and input messages s_0 and s_1 of the sender we require that, if $(\mathsf{otm}_1, \omega) \leftarrow \mathsf{OT}_1(1^\lambda, \beta)$, $\mathsf{otm}_2 \leftarrow \mathsf{OT}_2(\mathsf{otm}_1, (s_0, s_1))$, then $\mathsf{OT}_3(\mathsf{otm}_2, (\beta, \omega)) = s_\beta$ with overwhelming probability.
- **Receiver's security.** We require that

$$\left\{\mathsf{otm}_1 : (\mathsf{otm}_1, \omega) \leftarrow \mathsf{OT}_1(1^\lambda, 0)\right\} \stackrel{c}{\approx} \left\{\mathsf{otm}_1 : (\mathsf{otm}_1, \omega) \leftarrow \mathsf{OT}_1(1^\lambda, 1)\right\}.$$

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

$$\left\{\beta, \omega', \mathsf{OT}_2(1^\lambda, \mathsf{otm}_1, K_0, K_1)\right\} \stackrel{c}{\approx} \left\{\beta, \omega', \mathsf{OT}_2(1^\lambda, \mathsf{otm}_1, L_0, L_1)\right\}$$

  where $\omega' \leftarrow \{0,1\}^*$ and $(\mathsf{otm}_1, \omega) := \mathsf{OT}_1(1^\lambda, \beta; \omega')$.

Constructions of semi-honest two-round OT are known in the plain model under assumptions such as CDH [BM89], DDH [AIR01; NP01] and quadratic/N^{th} residuosity [HK12].

Maliciously Secure Two-Round Oblivious Transfer. We consider the stronger notion of oblivious transfer in the common random/reference string model. In terms of syntax, we supplement the syntax of semi-honest oblivious transfer with an algorithm K_{OT} that takes the security parameter 1^λ as input and outputs the common random/reference string σ. Also, the three algorithms $\mathsf{OT}_1, \mathsf{OT}_2$ and OT_3 additionally take σ as input. Correctness and receiver's security properties in the malicious case are the same as the semi-honest case. However, we strengthen the sender's security as described below.

[2] We note that ω in the output of OT_1 need not contain all the random coins used by OT_1. This fact will be useful in the stronger equivocal security notion of oblivious transfer.

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

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

$$\left\{(\sigma, \mathsf{otm}_1) : \sigma \leftarrow K_{\mathsf{OT}}(1^\lambda), (\mathsf{otm}_1, \omega) \leftarrow OT_1(\sigma, 0)\right\} \stackrel{c}{\approx} \left\{(\sigma, \mathsf{otm}_1) : \sigma \leftarrow K_{\mathsf{OT}}(1^\lambda), (\mathsf{otm}_1, \omega) \leftarrow OT_1(\sigma, 1)\right\}$$

- **Sender's security.** We require the existence of PPT algorithm $\mathsf{Ext} = (\mathsf{Ext}_1, \mathsf{Ext}_2)$ such that for any choice of $K_0, K_1 \in \{0,1\}^\lambda$ and PPT adversary $\mathcal{A}$ we have that

$$\left|\Pr[\mathrm{IND}^{\mathrm{REAL}}_{\mathcal{A}}(1^\lambda, K_0, K_1) = 1] - \Pr[\mathrm{IND}^{\mathrm{IDEAL}}_{\mathcal{A}}(1^\lambda, K_0, K_1) = 1]\right| \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Experiment $\mathrm{IND}^{\mathrm{REAL}}_{\mathcal{A}}(1^\lambda, K_0, K_1)$:	**Experiment** $\mathrm{IND}^{\mathrm{IDEAL}}_{\mathcal{A}}(1^\lambda, K_0, K_1)$:
$\sigma \leftarrow K_{\mathsf{OT}}(1^\lambda)$	$(\sigma, \tau) \leftarrow \mathsf{Ext}_1(1^\lambda)$
$\mathsf{otm}_1 \leftarrow \mathcal{A}(\sigma)$	$\mathsf{otm}_1 \leftarrow \mathcal{A}(\sigma)$
	$\beta := \mathsf{Ext}_2(\tau, \mathsf{otm}_1)$
	$L_0 := K_\beta$ and $L_1 := K_\beta$
$\mathsf{otm}_2 \leftarrow \mathsf{OT}_1(\sigma, \mathsf{otm}_1, (K_0, K_1))$	$\mathsf{otm}_2 \leftarrow \mathsf{OT}_2(\sigma, \mathsf{otm}_1, (L_0, L_1))$
Output $\mathcal{A}(\mathsf{otm}_2)$	Output $\mathcal{A}(\mathsf{otm}_2)$

Constructions of maliciously secure two-round OT are known in the common random string model under assumptions such as DDH, quadratic residuosity, and LWE [PVW08].

Equivocal Receiver's Security. We also consider a strengthened notion of malicious receiver's security where we require the existence of a PPT simulator Sim_{Eq} such that the for any $\beta \in \{0,1\}$:

$$\left\{(\sigma, (\mathsf{otm}_1, \omega_\beta)) : (\sigma, \mathsf{otm}_1, \omega_0, \omega_1) \leftarrow \mathsf{Sim}_{Eq}(1^\lambda)\right\} \stackrel{c}{\approx} \left\{(\sigma, \mathsf{OT}_1(\sigma, \beta)) : \sigma \leftarrow K_{\mathsf{OT}}(1^\lambda)\right\}.$$

Using standard techniques in the literature (e.g., [Can+02]) it is possible to add equivocal receiver's security to any OT protocol. We give a construction in below for completeness.

Lemma 3.2.2 *Assuming two-round maliciously secure OT protocol, there exists a two-round maliciously secure OT protocol with equivocal receiver's security.*

Proof Given a two-round maliciously secure OT protocol $(K'_{\mathsf{OT}}, \mathsf{OT}'_1, \mathsf{OT}'_2, \mathsf{OT}'_3)$ we give a two-round maliciously secure OT protocol $(K_{\mathsf{OT}}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ that additionally achieves the equivocal receiver's security. We also use a pseudorandom generator $g : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$. Our construction is as follows:

- $K_{\mathsf{OT}}(1^\lambda)$: Output $\sigma := (\sigma', r)$ where $\sigma' \leftarrow K'_{\mathsf{OT}}(1^\lambda)$ and $r \leftarrow \{0,1\}^{3\lambda}$.
- $\mathsf{OT}_1(\sigma = (\sigma', r), \beta)$:
  1. Sample $x \leftarrow \{0,1\}^\lambda$. If $\beta = 0$ then set $y := g(x)$ and $y := r \oplus g(x)$ otherwise.
  2. For each $i \in [\lambda]$, prepare $(\mathsf{otm}^0_{1,i}, \omega^0_i) \leftarrow \mathsf{OT}'_1(\sigma', x_i)$.
  3. For each $i \in [\lambda]$, prepare $(\mathsf{otm}^1_{1,i}, \omega^1_i) \leftarrow \mathsf{OT}'_1(\sigma', x_i)$.
  4. Output $\mathsf{otm}_1 := (y, \{\mathsf{otm}^0_{1,i}, \mathsf{otm}^1_{1,i}\}_{i\in[\lambda]})$ and $\omega := \left(\beta, \{\omega^0_i\}_{i\in[\lambda]}\right)$ if $\beta = 0$ and $\omega := \left(\beta, \{\omega^1_i\}_{i\in[\lambda]}\right)$ otherwise.
- $\mathsf{OT}_2(\sigma = (\sigma', r), \mathsf{otm}_1 = (y, \{\mathsf{otm}^0_{1,i}, \mathsf{otm}^1_{1,i}\}_{i\in[\lambda]}), (s_0, s_1))$: Let $C_{y,s}$ be a circuit with $y \in \{0,1\}^{3\lambda}$ and s hardwired in it which on input $x \in \{0,1\}^\lambda$ outputs s if $y = g(x)$ and $\perp$ otherwise. OT_2 proceeds as follows:
  1. Obtain $(\widetilde{C}^0, \{\mathsf{lab}^0_{i,b}\}_{i\in[\lambda],b\in\{0,1\}}) \leftarrow \mathsf{Garb}(1^\lambda, C_{y,s_0})$.
  2. Obtain $(\widetilde{C}^1, \{\mathsf{lab}^1_{i,b}\}_{i\in[\lambda],b\in\{0,1\}}) \leftarrow \mathsf{Garb}(1^\lambda, C_{r\oplus y,s_1})$.
  3. For each $i \in [\lambda]$, obtain $\mathsf{otm}^0_{2,i} \leftarrow \mathsf{OT}'_2(\sigma', \mathsf{otm}^0_{1,i}, (\mathsf{lab}^0_{i,0}, \mathsf{lab}^0_{i,1}))$.
  4. For each $i \in [\lambda]$, obtain $\mathsf{otm}^1_{2,i} \leftarrow \mathsf{OT}'_2(\sigma', \mathsf{otm}^1_{1,i}, (\mathsf{lab}^1_{i,0}, \mathsf{lab}^1_{i,1}))$.
  5. Output $\mathsf{otm}_2 := (\widetilde{C}^0, \widetilde{C}^1, \{\mathsf{otm}^0_{2,i}, \mathsf{otm}^1_{2,i}\}_{i\in[\lambda]})$.
- $\mathsf{OT}_3\left(\sigma = (\sigma', r), \mathsf{otm}_2 = (\widetilde{C}^0, \widetilde{C}^1, \{\mathsf{otm}^0_{2,i}, \mathsf{otm}^1_{2,i}\}_{i\in[\lambda]}), \omega = \left(\beta, \left\{\omega^\beta_i\right\}_{i\in[\lambda]}\right)\right)$: Compute
  1. For each $i \in [\lambda]$, recover $\mathsf{lab}_i := \mathsf{OT}'_3(\sigma', \mathsf{otm}^\beta_{2,i}, \omega^\beta_i)$.
  2. Output $\mathsf{Eval}(\widetilde{C}^\beta, \{\mathsf{lab}_i\}_{i\in[\lambda]})$.

The correctness of the above described OT protocol follows directly from the correctness of the underlying cryptographic primitives. We now prove sender security and equivocal receiver's security.

Sender's Security. The sender's security of $(K_{\mathsf{OT}}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ follows from the sender's security of $(K'_{\mathsf{OT}}, \mathsf{OT}'_1, \mathsf{OT}'_2, \mathsf{OT}'_3)$ and the simulation security of the garbling scheme. We start by describing the construction of $\mathsf{Ext} = (\mathsf{Ext}_1, \mathsf{Ext}_2)$ using the extractor $\mathsf{Ext}' = (\mathsf{Ext}'_1, \mathsf{Ext}'_2)$ for $(K'_{\mathsf{OT}}, \mathsf{OT}'_1, \mathsf{OT}'_2, \mathsf{OT}'_3)$.

- $\mathsf{Ext}_1(1^\lambda)$ executes $(\sigma', \tau) \leftarrow \mathsf{Ext}'_1(1^\lambda)$ and $r \leftarrow \{0,1\}^{3\lambda}$ and outputs $\sigma := (\sigma', r)$ and τ.
- $\mathsf{Ext}_2\left(\tau, \mathsf{otm}_1 = \left(y, \left\{\mathsf{otm}^0_{1,i}, \mathsf{otm}^1_{1,i}\right\}_{i\in[\lambda]}\right)\right)$ proceeds as follows: For each $i \in [\lambda]$, obtain $x_{0,i} := \mathsf{Ext}'_2(\tau, \mathsf{otm}_{1,i})$. If $g(x_0) = y$ then output 0 and 1 otherwise.

Now we argue that using this extractor $\mathsf{Ext} = (\mathsf{Ext}_1, \mathsf{Ext}_2)$, for any PPT adversary $\mathcal{A}$, the distributions $\text{IND}^{\text{REAL}}_{\mathcal{A}}(1^\lambda, K_0, K_1)$ and $\text{IND}^{\text{IDEAL}}_{\mathcal{A}}(1^\lambda, K_0, K_1)$ are computationally indistinguishable. We argue this via the following sequence of hybrids.

- $\mathcal{H}_0$: This hybrid is the same as $\text{IND}^{\text{REAL}}_{\mathcal{A}}(1^\lambda, K_0, K_1)$.
- $\mathcal{H}_1$: In this hybrid we change how the σ' in $\sigma = (\sigma', r)$ is generated. Specifically, we use the extractor Ext'_1 above to generate it. Additionally, we use Ext'_2 to recover a value of x_0 and x_1 that the receiver provides in $\{\mathsf{otm}^0_{1,i}\}_{i\in[\lambda]}$ and $\{\mathsf{otm}^1_{1,i}\}_{i\in[\lambda]}$, respectively.

  Indistinguishability between $\mathcal{H}_0$ and $\mathcal{H}_1$ can be reduced directly to the sender's security of the underlying OT protocol. Additionally, we claim that for any x_0, x_1 over the random choices of r we have that $\Pr[g(x_0) = y \wedge g(x_1) = r \oplus y] \leq 2^{-\lambda}$ which is negligible. Thus, we set $\beta = 0$ if $g(x_0) = y$ and 1 otherwise. This is the same as the value extracted by Ext_2 above.
- $\mathcal{H}_2$: In this hybrid we change how the values $\mathsf{otm}^{1-\beta}_{2,i}$ are generated for each $i \in [\lambda]$. More specifically, for each $i \in [\lambda]$, we generate $\mathsf{otm}^{1-\beta}_{2,i} \leftarrow \mathsf{OT}_2\left(\sigma', \mathsf{otm}^{1-\beta}_{1,i}, (\mathsf{lab}^{1-\beta}_{i,x_{1-\beta,i}}, \mathsf{lab}^{1-\beta}_{i,x_{1-\beta,i}})\right)$.

  Indistinguishability between $\mathcal{H}_1$ and $\mathcal{H}_2$ can be reduced to the sender's security of the underlying OT protocol.
- $\mathcal{H}_3$: In this hybrid we change the garbled $\widetilde{C}^{1-\beta}$ to the simulate circuit generated via $\mathsf{Sim}_{\mathsf{G}}$ with the output $\perp$ hardwired (i.e. it is generated as $\mathsf{Sim}_{\mathsf{G}}(1^\lambda, \perp)$).

  Indistinguishability between $\mathcal{H}_2$ and $\mathcal{H}_3$ reduces to the security of the garbling scheme.

Equivocal Receiver's Security. We start by providing the PPT simulator $\mathsf{Sim}_{Eq}(1^\lambda)$ which proceeds as follows:

1. Generate $\sigma' \leftarrow K'_{\mathsf{OT}}(1^\lambda)$ and $r := g(x_0) \oplus g(x_1)$ where $x_0, x_1 \leftarrow \{0,1\}^\lambda$. Set $\sigma := (\sigma', r)$.
2. Set $y := g(x_0)$.
3. For each $i \in [\lambda]$, prepare $(\mathsf{otm}^0_{1,i}, \omega^0_i) \leftarrow \mathsf{OT}'_1(\sigma', x_{0,i})$.
4. For each $i \in [\lambda]$, prepare $(\mathsf{otm}^1_{1,i}, \omega^1_i) \leftarrow \mathsf{OT}'_1(\sigma', x_{1,i})$.
5. Output $\left(\sigma := (\sigma', r), \mathsf{otm}_1 := (y, \{\mathsf{otm}^0_{1,i}, \mathsf{otm}^1_{1,i}\}_{i\in[\lambda]}), \omega_0 := \left(\beta, \{\omega^0_i\}_{i\in[\lambda]}\right), \omega_1 := \left(\beta, \{\omega^1_i\}_{i\in[\lambda]}\right)\right)$.

We are left to argue that for each β, the distribution $(\sigma, \mathsf{otm}_1, \omega_\beta)$ is indistinguishable for the distribution of the honestly generated values. We sketch the argument for the case where $\beta = 0$. The argument for the case where $\beta = 1$ is analogous.

- $\mathcal{H}_0$: This hybrid corresponds to the real distribution. Namely, we set $\sigma = (\sigma', r) \leftarrow K_{\mathsf{OT}}(1^\lambda)$ and $(\mathsf{otm}_1 = (y, \{\mathsf{otm}^0_{1,i}, \mathsf{otm}^1_{1,i}\}_{i\in[\lambda]}), \omega_\beta) \leftarrow \mathsf{OT}_1(\sigma, \beta)$.

- $\mathcal{H}_1$: In this hybrid, we change how r in generated. More specifically, we set r as $g(x) \oplus g(x')$ where $x, x' \leftarrow \{0,1\}^\lambda$ and use the same x in the generation of otm_1.

  Indistinguishability between hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ follows directly from the security of the pseudorandom generator.

- $\mathcal{H}_2$: In this hybrid we change how $\mathsf{otm}^1_{1,i}$ values are generated. Specifically, for each $i \in [\lambda]$, we set $(\mathsf{otm}^1_{1,i}, \omega^1_i) \leftarrow \mathsf{OT}'_1(\sigma', x'_i)$ instead of $(\mathsf{otm}^1_{1,i}, \omega^1_i) \leftarrow \mathsf{OT}'_1(\sigma', x_i)$. Note that $\mathcal{H}_2$ is the same as the distribution generated by Sim_{Eq} for $\beta = 0$ case.

  Indistinguishability between hybrids $\mathcal{H}_1$ and $\mathcal{H}_2$ follows from the receiver's security of the underlying OT protocol.

This completes the argument. ■

3.3 Conforming Protocols

Our result is obtained by transforming larger-round protocols securely computing a function and satisfying certain syntactic structure into a two-round protocol that securely computes the same function. We refer to protocols satisfying this syntax as *conforming protocols*. In this subsection, we describe this notion and prove that any MPC protocol can be transformed into a conforming protocol while preserving its correctness and security properties.

3.3.1 Specifications for a Conforming Protocol

Consider an n party deterministic[3] MPC protocol Φ between parties $P_1, \ldots, P_n$ with inputs $x_1, \ldots, x_n$, respectively. For each $i \in [n]$, we let $x_i \in \{0,1\}^m$ denote the input of party P_i. A conforming protocol Φ is defined by functions pre, post, and computations steps or what we call *actions* $\phi_1, \cdots \phi_T$. The protocol Φ proceeds in three stages: the pre-processing stage, the computation stage and the output stage.

- **Pre-processing phase**: For each $i \in [n]$, party P_i computes

  $$(z_i, v_i) \leftarrow \mathsf{pre}(1^\lambda, i, x_i)$$

  where pre is a randomized algorithm. The algorithm pre takes as input the index i of the party, its input x_i and outputs $z_i \in \{0,1\}^{\ell/n}$ and $v_i \in \{0,1\}^\ell$ (where ℓ is a parameter of the protocol). Finally, 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 $k \in [\ell] \setminus \{(i-1)\ell/n + 1, \ldots, i\ell/n\}$.

- **Computation phase**: For each $i \in [n]$, party P_i sets

  $$\mathsf{st}_i := (z_1 \| \cdots \| z_n) \oplus v_i.$$

  Next, for each $t \in \{1 \cdots T\}$ parties proceed as follows:

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

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

$$\mathsf{st}_{i,h} = \mathsf{NAND}(\mathsf{st}_{i,f}, \mathsf{st}_{i,g})$$

   and broadcasts $\mathsf{st}_{i,h} \oplus v_{i,h}$ to every other party.
3. Every party P_j for $j \neq i$ updates $\mathsf{st}_{j,h}$ to the bit value received from P_i.

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

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

3.3.2 Transformation for Making a Protocol Conforming

We show that any MPC protocol can made conforming by making only some syntactic changes. Our transformed protocols retains the correctness or security properties of the original protocol.

Lemma 3.3.1 *Any MPC protocol Π can be transformed to a conforming protocol Φ while inheriting the correctness and the security of the original protocol.*

Proof Let Π be any given MPC protocol. Without loss of generality we assume that in each round of Π, *one* party broadcasts *one* bit that is obtained by computing a circuit on its initial state and the messages it has received so far from other parties. Note that this restriction can be easily enforced by increasing the round complexity of the protocol to the communication complexity of the protocol. Let the round complexity (and also communication complexity) of Π be p. In every round $r \in [p]$ of Π, a single bit is sent by one of the parties by computing a circuit on its internal state. Let the circuit computed in round r be C_r. Without loss of generality we assume that (i) these exists q such that for each $r \in [p]$, we have that $q = |C_r|$, (ii) each C_r is composed of just NAND gates with fan-in two, and (iii) each party sends an equal number of bits in the execution of Π. All three of these conditions can be met by adding dummy gates and dummy round of interaction.

We are now ready to describe our transformed conforming protocol Φ. The protocol Φ will have $T = pq$ rounds. We let $\ell = mn + pq$ and $\ell' = pq/n$ and depending on ℓ the compiled protocol Φ is as follows.

- $\mathsf{pre}(i, x_i)$: Sample $r_i \leftarrow \{0,1\}^m$ and $s_i \leftarrow (\{0,1\}^{q-1} \| 0)^{p/n}$. (Observe that s_i is a pq/n bit random string such that its $q^{th}, 2q^{th} \cdots$ locations are set to 0.) Output $z_i := x_i \oplus r_i \| 0^{\ell'}$ and $v_i := 0^{\ell/n} \| \ldots \| r_i \| s_i \| \ldots \| 0^{\ell/n}$.

- We are now ready to describe the actions $\phi_1, \cdots \phi_T$. For each $r \in [p]$, round r in Π party is expanded into q actions in Φ — namely, actions $\{\phi_j\}_j$ where $j \in \{(r-1)q+1 \cdots rq\}$. Let P_i be the party that computes the circuit C_r and broadcast the output bit broadcast in round r of Π. We now describe the ϕ_j for $j \in \{(r-1)q+1 \cdots rq\}$. For each j, we set $\phi_j = (i, f, g, h)$ where f and g are the locations in st_i that the j^{th} gate of C_r is computed on (recall that initially st_i is set to $z_i \oplus v_i$). Moreover, we set h to be the first location in st_i among the locations $(i-1)\ell/n + m + 1$ to $i\ell/n$ that has previously not been assigned to an action. (Note that this is ℓ' locations which is exactly equal to the number of bits computed and broadcast by P_i.)

  Recall from before than on the execution of ϕ_j, party P_i sets $\mathsf{st}_{i,h} := \mathsf{NAND}(\mathsf{st}_{i,f}, \mathsf{st}_{i,g})$ and broadcasts $\mathsf{st}_{i,h} \oplus v_{i,h}$ to all parties.

- $\mathsf{post}(i, \mathsf{st}_i)$: Gather the local state of P_i and the messages sent by the other parties in Π from st_i and output the output of Π.

Now we need to argue that Φ preserves the correctness and security properties of Π. Observe that Φ is essentially the same as the protocol Π except that in Φ some additional bits are sent. Specifically, in addition to the messages that were sent in Π, in Φ parties send z_i in the preprocessing step and $q-1$ additional bits per every bit sent in Π. Note that these additional bits sent are not used in the computation of Φ. Thus these bits do not affect the functionality of Π if dropped. This ensures that Φ inherits the correctness properties of Π. Next note that each of these bits is masked by a uniform independent bit. This ensures that Φ achieves the same security properties as the underlying properties of Π.

Finally, note that by construction 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 $h \neq h'$ as required. ■

3.4 Two-Round MPC: Semi-Honest Case

In this section, we give our construction of two-round multiparty computation protocol in the semi-honest case with security against static corruptions based on any two-round semi-honest oblivious transfer protocol in the plain model. This is achieved by designing a compiler that takes any conforming arbitrary (polynomial) round MPC protocol Φ and squishes it to two rounds.

3.4.1 Our Compiler

We give our construction of two-round MPC in Figure 3.3 and the circuit that needs to be garbled (repeatedly) is shown in Figure 3.4. We start by providing intuition behind this construction.

3.4.1.1 Intuition

In the first round of our compiled protocol, each party runs the preprocessing phase of the protocol Φ and obtains z_i and v_i and broadcasts z_i to every other party. In the second round, each party sends a set of garbled circuits that "non-interactively" implement the entire computation phase of the protocol Φ. In other words, any party with the set of garbled circuits sent by every other party, can use them to compute the entire transcript of the computation phase of the protocol Φ. This allows each party to obtain the output of the protocol Φ. In the following paragraphs, we give more details on how this is achieved.

To understand the main idea, let us concentrate on a particular round (let us say the t^{th} round) of the computation phase of the conforming protocol Φ and see how this step is implemented using garbled circuits. Recall that before starting the computation phase, each party locally computes $\mathsf{st}_i := (z_1 \| \ldots \| z_n) \oplus v_i$ using the first round messages sent by the other parties. This local state is updated (recall that only one bit location is updated) at the end of each round based on the bit that is sent in that round. We start with some notations.

Notations. Let us say that the party P_{i^*} is the designated party in round t. Let st_i^t be the updated local state of party P_i at the beginning of the t^{th} round of the computation phase. In the t^{th} round, the designated party P_{i^*} computes $\gamma := \mathsf{NAND}(\mathsf{st}_{i^*,f}^t, \mathsf{st}_{i^*,g}^t)$, writes this bit to position h of $\mathsf{st}_{i^*}^t$ and broadcasts $\gamma \oplus v_{i^*,h}$ to every other party. Every other party P_i (where $i \neq i^*$) updates its local state by writing the received bit at position h in its state st_i^t.

Implementing the Computation Phase. The t^{th} round of the computation phase is implemented by the t^{th} garbled circuit in each of these sequences. In a bit more details, the garbled circuit of party P_i takes as input st_i^t which is the state of the party P_i at the beginning of the t-th round and outputs or, aids the process of outputting the labels corresponding to the updated local state at the end of the t^{th} round. These labels are then used to evaluate the garbled circuit corresponding to the $(t+1)^{th}$ round of the computation phase and this process continues. Finally, at the end each party can just compute output function on the final local state to obtain its output. Next, we describe how the t^{th} sequence of garbled circuits can be used to complete the t^{th} action of the computation phase.

The t^{th} garbled circuit of party P_{i^*} is executed first and is the most natural one as in this round party P_{i^*} is the one that sends a bit to the other parties. Starting with the easy part, this garbled circuit takes as input $\mathsf{st}_{i^*}^t$, updates the local state by writing the bit γ in the position h of $\mathsf{st}_{i^*}^t$ and outputs the labels corresponding to its updated state. However, the main challenge is that this garbled circuit needs to communicate the bit $\gamma \oplus v_{i^*,h}$ to other garbled circuits of the other parties. Specifically, those garbled circuits also need to output the correct labels corresponding to the their updated local state. Note that only the h^{th} bit of each of their local state needs to be updated.

Relying on Oblivious Transfer. In addition to broadcasting the encoded input z_i in the first round, the party P_i sends a set of 4 OT messages (acting as the receiver) for every round in the computation phase where P_i is the designated party. Thus, if the number of rounds in the computation phase where P_i is the designated party is a_i, then the party P_i sends $4a_i$ receiver OT messages. Specifically, in our running example from above, P_{i^*} will generate 4 first OT messages to help in t^{th} round of Φ. In particular, for each value of $\alpha, \beta \in \{0,1\}$, P_{i^*} generates the first OT message with $v_{i^*,h} \oplus \mathsf{NAND} v_{i^*,f} \oplus \alpha, v_{i^*,g} \oplus \beta$ as its choice bit. Every other party P_i for $i \neq i^*$ acts as the sender and prepares four OT responses corresponding to each of the four OT messages using labels corresponding to the h-th input wire (say $(\mathsf{lab}_{h,0}^{i,t+1}, \mathsf{lab}_{h,1}^{i,t+1})$) of its next (i.e., $(t+1)^{th}$) garbled circuit. However, these values aren't sent to anyone yet! Because sending them all to P_{i^*} would lead to complete loss of security. Specifically, for every choice of $v_{i^*,f}, v_{i^*,g}, v_{i^*,h}$ there exists different choices of α, β such that $v_{i^*,h} \oplus \mathsf{NAND} v_{i^*,f} \oplus \alpha, v_{i^*,g} \oplus \beta$ is 0 and 1, respectively. Thus, if all these OT responses were reveled to P_{i^*} then P_{i^*} would learn both the input labels $\mathsf{lab}_{h,0}^{i,t+1}, \mathsf{lab}_{h,1}^{i,t+1}$ potentially breaking the security of garbled circuits. Our key idea here is that party P_i hardcodes these OT responses in its t^{th} garbled circuit and only one of them is revealed to P_{i^*}. We now elaborate this.

The t-th garbled circuit of party P_i (where $i \neq i^*$) outputs the set of labels corresponding to the state bits $\{\mathsf{st}_{i,k}^t\}_{k \in [\ell] \setminus \{h\}}$ (as these bits do not change at the end of the t-th round) and additionally outputs the sender OT response for $\alpha = \mathsf{st}_{i,f}^t$ and $\beta = \mathsf{st}_{i,g}^t$ with the messages being set to the labels corresponding to h-th bit of st_i^t. It follows from the invariant of the protocol, that the choice bit in this OT_1 message is indeed $\gamma \oplus v_{i^*,h}$ which is exactly the bit P_{i^*} wants to communicate to the other parties. However, this leaves us with another problem. The OT responses only allow P_{i^*} to learn the labels of the next garbled circuits and it is unclear how a party $j \neq i^*$ obtains the labels of the garbled circuits generated by P_i.

Enabling all Parties to Compute. The party P_{i^*}'s t^{th} garbled circuit, in addition to outputting the labels corresponding to the updated state of P_{i^*}, outputs the randomness it used to prepare the first OT message for which all P_i for $i \neq i^*$ output OT responses; namely, $\alpha = \mathsf{st}_{i^*,f}^t \oplus v_{i^*,f}, \beta = \mathsf{st}_{i^*,g}^t \oplus v_{i^*,g}$. It again follows from the invariant of the protocol Φ that this allows every party P_j with $j \neq i^*$ to evaluate the recover $\mathsf{lab}_{h,\gamma \oplus v_{i^*,h}}^{i,t+1}$ which is indeed the label corresponding to the correct updated state. Thus, using the randomness output by the garbled circuit of P_{i^*} all other parties can recover the label $\mathsf{lab}_{h,\gamma \oplus v_{i^*,h}}^{i,t+1}$.

We stress that this process of revealing the randomness of the OT leads to complete loss of security for the particular instance OT. Nevertheless, since the randomness of only one of the four OT messages of P_{i^*} is reveled, overall security is ensured. In particular, our construction ensures that the learned choice bit is $\gamma \oplus v_{i^*,h}$ which is in fact the message that is broadcasted in the underlying protocol Φ. Thus, it follows from the security of the protocol Φ that learning this message does not cause any vulnerabilities.

Let Φ be an n-party conforming semi-honest MPC protocol, $(\mathsf{Garb}, \mathsf{Eval})$ be a garbling scheme for circuits and $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ be a semi-honest two-round oblivious transfer protocol.

Round-1: Each party P_i does the following:

1. Compute $(z_i, v_i) \leftarrow \mathsf{pre}(1^\lambda, i, x_i)$.
2. For each t such that $\phi_t = (i, f, g, h)$ (A_i is the set of such values of t), for each $\alpha, \beta \in \{0,1\}$, choose $\omega_{t,\alpha,\beta}$ uniformly from $\{0,1\}^*$ and compute
$$\mathsf{otm}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}_1(1^\lambda, v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta); \omega_{t,\alpha,\beta}).$$
3. Send $\left(z_i, \{\mathsf{otm}_{1,t,\alpha,\beta}\}_{t \in A_i, \alpha, \beta \in \{0,1\}}\right)$ to every other party.

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

1. Set $\mathsf{st}_i := (z_1 \| \ldots \| z_{i-1} \| z_i \| z_{i+1} \| \ldots \| z_n) \oplus v_i$.
2. Set $\overline{\mathsf{lab}}^{i,T+1} := \{\mathsf{lab}_{k,0}^{i,T+1}, \mathsf{lab}_{k,1}^{i,T+1}\}_{k \in [\ell]}$ where for each $k \in [\ell]$ and $b \in \{0,1\}$ $\mathsf{lab}_{k,b}^{i,T+1} := 0^\lambda$.
3. **for** each t from T down to 1,
   a) Parse ϕ_t as (i^*, f, g, h).
   b) If $i = i^*$ then compute (where P is described in Figure 3.4)
   $$\left(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{lab}}^{i,t}\right) \leftarrow \mathsf{Garb}(1^\lambda, \mathsf{P}[i, \phi_t, v_i, \{\omega_{t,\alpha,\beta}\}_{\alpha,\beta}, \bot, \overline{\mathsf{lab}}^{i,t+1}]).$$
   c) If $i \neq i^*$ then for every $\alpha, \beta \in \{0,1\}$, set $\mathsf{otm}_{2,t,\alpha,\beta}^i \leftarrow \mathsf{OT}_2(\mathsf{otm}_{1,t,\alpha,\beta}, \mathsf{lab}_{h,0}^{i,t+1}, \mathsf{lab}_{h,1}^{i,t+1})$ and compute
   $$\left(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{lab}}^{i,t}\right) \leftarrow \mathsf{Garb}(1^\lambda, \mathsf{P}[i, \phi_t, v_i, \bot, \{\mathsf{otm}_{2,t,\alpha,\beta}^i\}_{\alpha,\beta}, \overline{\mathsf{lab}}^{i,t+1}]).$$
4. Parse $\overline{\mathsf{lab}}^{i,1}$ as $\{\mathsf{lab}_{k,0}^{i,1}, \mathsf{lab}_{k,1}^{i,1}\}_{k \in [\ell]}$ and send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t \in [T]}, \{\mathsf{lab}_{k,\mathsf{st}_{i,k}}^{i,1}\}_{k \in [\ell]}\right)$ to every other party.

Evaluation: To compute the output of the protocol, each party P_i does the following:

1. For each $j \in [n]$, let $\widetilde{\mathsf{lab}}^{j,1} := \{\mathsf{lab}_k^{j,1}\}_{k \in [\ell]}$ be the labels received from party P_j at the end of round 2.
2. **for** each t from 1 to T do:
   a) Parse ϕ_t as (i^*, f, g, h).
   b) Compute $((\alpha, \beta, \gamma), \omega, \widetilde{\mathsf{lab}}^{i^*,t+1}) := \mathsf{Eval}(\widetilde{\mathsf{P}}^{i^*,t}, \widetilde{\mathsf{lab}}^{i^*,t})$.
   c) Set $\mathsf{st}_{i,h} := \gamma \oplus v_{i,h}$.
   d) **for** each $j \neq i^*$ do:
      i. Compute $(\mathsf{otm}_2, \{\mathsf{lab}_k^{j,t+1}\}_{k \in [\ell] \setminus \{h\}}) := \mathsf{Eval}(\widetilde{\mathsf{P}}^{j,t}, \widetilde{\mathsf{lab}}^{j,t})$.
      ii. Recover $\mathsf{lab}_h^{j,t+1} := \mathsf{OT}_3(\mathsf{otm}_2, \omega)$.
      iii. Set $\widetilde{\mathsf{lab}}^{j,t+1} := \{\mathsf{lab}_k^{j,t+1}\}_{k \in [\ell]}$.
3. Compute the output as $\mathsf{post}(\mathsf{st}_i)$.

Figure 3.3: Two-round Semi-Honest MPC.

P

Input. st_i.
Hardcoded. The index i of the party, the action $\phi_t = (i^*, f, g, h)$, the secret value v_i, the strings $\{\omega_{t,\alpha,\beta}\}_{\alpha,\beta}$, $\{\mathsf{otm}_{2,t,\alpha,\beta}\}_{\alpha,\beta}$ and a set of labels $\overline{\mathsf{lab}} = \{\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}\}_{k\in[\ell]}$.

1. **if** $i = i^*$ **then**:
   a) Compute $\mathsf{st}_{i,h} := \mathsf{NAND}(\mathsf{st}_{i,f}, \mathsf{st}_{i,g})$, $\alpha := \mathsf{st}_{i,f} \oplus v_{i,f}$, $\beta := \mathsf{st}_{i,g} \oplus v_{i,g}$ and $\gamma := \mathsf{st}_{i,h} \oplus v_{i,h}$.
   b) Output $((\alpha, \beta, \gamma), \omega_{t,\alpha,\beta}, \{\mathsf{lab}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]})$.
2. **else**:
   a) Output $(\mathsf{otm}_{2,t,\mathsf{st}_{i,f},\mathsf{st}_{i,g}}, \{\mathsf{lab}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]\setminus\{h\}})$.

Figure 3.4: The program P.

Theorem 3.4.1 *Let Φ be a polynomial round, conforming, n-party semi-honest MPC protocol computing a function $f : (\{0,1\}^m)^n \to \{0,1\}^*$, $(\mathsf{Garb}, \mathsf{Eval})$ be a garbling scheme for circuits, and $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ be a semi-honest two-round OT protocol. The protocol described in Figure 3.3 is a two-round, n-party semi-honest MPC protocol computing f against static corruptions.*

3.4.2 Correctness

In order to prove correctness, it is sufficient to show that the label computed in Step 2.(d).(ii) of the evaluation procedure corresponds to the bit $\mathsf{NAND}(\mathsf{st}_{i^*,f}, \mathsf{st}_{i^*,g}) \oplus v_{i^*,h}$. We show this by induction. Notice that by the assumption on the structure of v_{i^*} (recall that v_{i^*} is such that $v_{i^*,k} = 0$ for all $k \in [\ell]\setminus\{(i^*-1)\ell/n+1, \ldots, i^*\ell/n\}$) we deduce that for every $i \neq i^*$, $\mathsf{st}_{i,f} = \mathsf{st}_{i^*,f} \oplus v_{i^*,f}$ and $\mathsf{st}_{i,g} = \mathsf{st}_{i^*,g} \oplus v_{i^*,g}$ (from induction). Thus, the label obtained by OT_2 corresponds to the bit $\mathsf{NAND}(v_{i^*,f} \oplus \underbrace{\mathsf{st}_{i^*,f} \oplus v_{i^*,f}}_{\alpha}, v_{i^*,g} \oplus \underbrace{\mathsf{st}_{i^*,g} \oplus v_{i^*,g}}_{\beta}) \oplus v_{i^*,h} =$ $\mathsf{NAND}(\mathsf{st}_{i^*,f}, \mathsf{st}_{i^*,g}) \oplus v_{i^*,h}$ and correctness follows.

3.4.3 Simulator

Let $\mathcal{A}$ be a semi-honest adversary corrupting a subset of parties and let $H \subseteq [n]$ be the set of honest/uncorrupted parties. Since we assume that the adversary is static, this set is fixed before the execution of the protocol. Below we provide the description of our simulator.

Description of the Simulator. We give the description of the ideal world adversary S that simulates the view of the real world adversary $\mathcal{A}$. S will internally use the semi-honest simulator Sim_Φ for Φ, and the simulator Sim_G for garbling scheme for circuits. Recall that $\mathcal{A}$ is static and hence the set of honest parties H is known before the execution of the protocol.

Simulating the interaction with $\mathcal{Z}$. For every input value for the set of corrupted parties that S receives from $\mathcal{Z}$, S writes that value to $\mathcal{A}$'s input tape. Similarly, the output of $\mathcal{A}$ is written as the output on S's output tape.

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

- **Initialization**: S uses the inputs of the corrupted parties $\{x_i\}_{i\notin H}$ and output y of the functionality f to generate a simulated view of the adversary.[4] More formally, for each $i \in [n]\backslash H$, S sends $(\mathsf{input}, \mathsf{sid}, \{P_1 \cdots P_n\}, P_i, x_i)$ to the ideal functionality implementing f and obtains the output y. Next, it executes $\mathsf{Sim}_\Phi(1^\lambda, \{x_i\}_{i\notin H}, y)$ to obtain $\{z_i\}_{i\in H}$, the random tapes for the corrupted parties, the transcript of the computation phase denoted by $\mathsf{Z} \in \{0,1\}^t$ where Z_t is the bit sent in the t^{th} round of the computation phase of Φ, and the value st^* (which is equal to $\mathsf{st}_i^T \oplus v_i$ for each $i \in [n]$).[5] S starts the real-world adversary $\mathcal{A}$ with the inputs $\{z_i\}_{i\in H}$ and random tape generated by Sim_Φ.

- **Round-1 messages from S to $\mathcal{A}$:** Next S generates the OT messages on behalf of honest parties as follows. For each $i \in H, t \in A_i, \alpha, \beta \in \{0,1\}$, generate $\mathsf{otm}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}_1(1^\lambda, \mathsf{Z}_t; \omega_{t,\alpha,\beta})$ (for uniformly chosen $\omega_{t,\alpha,\beta}$). For each $i \in H$, S sends $(z_i, \{\mathsf{otm}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}})$ to the adversary $\mathcal{A}$ on behalf of the honest party P_i.

- **Round-1 messages from $\mathcal{A}$ to S:** Corresponding to every $i \in [n] \setminus H$, S receives from the adversary $\mathcal{A}$ the value $(z_i, \{\mathsf{otm}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}})$ on behalf of the corrupted party P_i.

- **Round-2 messages from S to $\mathcal{A}$:** For each $i \in H$, the simulator S generates the second round message on behalf of party P_i as follows:

  1. For each $k \in [\ell]$ set $\mathsf{lab}_k^{i,T+1} := 0^\lambda$.
  2. **for** each t from T down to 1,
     a) Parse ϕ_t as (i^*, f, g, h).
     b) Set $\alpha^* := \mathsf{st}_f^*$, $\beta^* := \mathsf{st}_g^*$, and $\gamma^* := \mathsf{st}_h^*$.
     c) If $i = i^*$ then compute

$$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}_k^{i,t}\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_\mathsf{G}\left(1^\lambda, 1^{|\mathsf{P}|}, 1^\ell, \left((\alpha^*, \beta^*, \gamma^*), \omega_{t,\alpha^*,\beta^*}, \{\mathsf{lab}_k^{i,t+1}\}_{k\in[\ell]}\right)\right).$$

[4] The case where each of the parties get a different output can be reduced to the case where the parties get the same output by XORing the outputs of the parties using a one-time pad.

[5] Note that for each $i \in [n]$, this value st^* is going to be the same

d) If $i \neq i^*$ then set $\mathsf{otm}^i_{2,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2(\mathsf{otm}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_h, \mathsf{lab}^{i,t+1}_h)$ and compute

$$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^\lambda, 1^{|\mathsf{P}|}, 1^\ell, \left(\mathsf{otm}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_k\}_{k\in[\ell]\setminus\{h\}}\right)\right).$$

3. Send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \{\mathsf{lab}^{i,1}_k\}_{k\in[\ell]}\right)$ to every other party.

- **Round-2 messages from $\mathcal{A}$ to S:** For every $i \in [n] \setminus H$, S obtains the second round message from $\mathcal{A}$ on behalf of the malicious parties. Subsequent to obtaining these messages, for each $i \in H$, S sends $(\mathsf{generateOutput}, \mathsf{sid}, \{P_1 \cdots P_n\})$ to the ideal functionality.

3.4.4 Proof of Indistinguishability

We now show that no environment $\mathcal{Z}$ can distinguish whether it is interacting with a real world adversary $\mathcal{A}$ or an ideal world adversary S. We prove this via an hybrid argument with $T+1$ hybrids.

- Hyb_{Real}: This hybrid is the same as the real world execution.

  Note that this hybrid is the same as hybrid Hyb_t below with $t = 0$.

- Hyb_t (where $t \in \{0, \ldots T\}$): Hybrid Hyb_t (for $t \in \{1 \cdots T\}$) is the same as hybrid Hyb_{t-1} except we change the distribution of the OT messages (both from the first and the second round of the protocol) and the garbled circuits (from the second round) that play a role in the execution of the t^{th} round of the protocol Φ; namely, the action $\phi_t = (i^*, f, g, h)$. We describe the changes more formally below.

  We start by executing the protocol Φ on the inputs and the random coins of the honest and the corrupted parties. This yields a transcript $\mathsf{Z} \in \{0,1\}^T$ of the computation phase. Since the adversary is assumed to be semi-honest, the execution of the protocol Φ with $\mathcal{A}$ will be consistent with Z. Let $\mathsf{st}^* = \mathsf{st}^T_i \oplus v_i$ for each $i \in [n]$. Finally, let $\alpha^* := \mathsf{st}^*_f$, $\beta^* := \mathsf{st}^*_g$ and $\gamma^* := \mathsf{st}^*_h$. For each $i \in H$, let st^{t+1}_i be the updated state of party P_i at the end of round t. In hybrid Hyb_t we make the following changes with respect to hybrid Hyb_{t-1}:

  - If $i^* \notin H$ then skip these changes. S makes two changes in how it generates messages on behalf of P_{i^*}. First, for all $\alpha, \beta \in \{0,1\}$, S generates $\mathsf{otm}_{1,t,\alpha,\beta}$ as $\mathsf{OT}_1(1^\lambda, \mathsf{Z}_t; \omega_{t,\alpha,\beta})$ (note that only one of these four values is subsequently used) rather than $\mathsf{OT}_1(1^\lambda, v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta); \omega_{t,\alpha,\beta})$. Second, it generates the garbled circuit

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

    where $\{\mathsf{lab}^{i^*,t+1}_{k,\mathsf{st}^{t+1}_{i,k}}\}_{k\in[\ell]}$ are the honestly generates input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i^*,t+1}$.

– S makes the following two changes in how it generates messages for other honest parties P_i (i.e., $i \in H \setminus \{i^*\}$). S does not generate four $\mathsf{otm}^i_{2,t,\alpha,\beta}$ values but just one of them; namely, S generates $\mathsf{otm}^i_{2,t,\alpha^*,\beta^*}$ as $\mathsf{OT}_2(\mathsf{otm}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t})$ rather than $\mathsf{OT}_2(\mathsf{otm}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$. Second it generates the garbled circuit

$$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^\lambda, 1^{|\mathsf{P}|}, 1^\ell, \left(\mathsf{otm}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^{t+1}_{i,k}}\}_{k\in[\ell]\setminus\{h\}}\right)\right),$$

where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^{t+1}_{i,k}}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i,t+1}$.

Indistinguishability between Hyb_{t-1} and Hyb_t is proved in Lemma 3.4.2.

- Hyb_{T+1}: In this hybrid we just change how the transcript Z, $\{z_i\}_{i\in H}$, random coins of malicious parties and value st^* are generated. Instead of generating these using honest party inputs, we generate it via the simulator Sim_Φ (of the semi-honest secure protocol Φ). In other words, we execute the simulator Sim_Φ on input $\{x_i\}_{i\in[n]\setminus H}$ and the output y obtained from the ideal functionality.

  The indistinguishability between hybrids Hyb_T and Hyb_{T+1} follows directly from the semi-honest security of the protocol Φ. Finally note that Hyb_{T+1} is same as the ideal execution (i.e., the simulator described in the previous subsection).

Lemma 3.4.2 *Assuming semi-honest security of the two-round OT protocol and the security of the garbling scheme, for all $t \in \{1 \dots T\}$ hybrids Hyb_{t-1} and Hyb_t are computationally indistinguishable.*

Proof Using the same notation as before, let $\phi_t = (i^*, f, g, h)$, $\mathsf{st}^{t+1}_{i^*}$ be the state of P_{i^*} at the end of round t, and $\alpha^* := \mathsf{st}^{t+1}_{i^*,f} \oplus v_{i^*,f}$, $\beta^* := \mathsf{st}^{t+1}_{i^*,g} \oplus v_{i^*,g}$ and $\gamma^* := \mathsf{st}^{t+1}_{i^*,h} \oplus v_{i^*,h}$. The indistinguishability between hybrids Hyb_{t-1} and Hyb_t follows by a sequence of three sub-hybrids $\mathsf{Hyb}_{t,1}$, $\mathsf{Hyb}_{t,2}$, and $\mathsf{Hyb}_{t,3}$.

- $\mathsf{Hyb}_{t,1}$: Hybrid $\mathsf{Hyb}_{t,1}$ is same as hybrid Hyb_{t-1} except that S now generates the garbled circuits $\widetilde{\mathsf{P}}^{i,t}$ for each $i \in H$ in a simulated manner (rather than generating them honestly). Specifically, instead of generating each garbled circuit and input labels $\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right)$ honestly, they are generated via the simulator by hard coding the output of the circuit itself. In a bit more details, parse ϕ_t as (i^*, f, g, h).

  – If $i = i^*$ then

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

  where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^{t+1}_{i,k}}\}_{k\in[\ell]}$ are the honestly generates input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i,t+1}$.

– If $i \neq i^*$ then

$$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}_k^{i,t}\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^\lambda, 1^{|\mathsf{P}|}, 1^\ell, \left(\mathsf{otm}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^{t+1}_{i,k}}\}_{k\in[\ell]\setminus\{h\}}\right)\right),$$

where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^{t+1}_{i,k}}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i,t+1}$.

The indistinguishability between hybrids $\mathsf{Hyb}_{t,1}$ and Hyb_{t-1} follows by $|H|$ invocations of security of the garbling scheme.

- $\mathsf{Hyb}_{t,2}$: Skip this hybrid if there does not exist $i \neq i^*$ such that $i \in H$. In this hybrid, we change how S generates the $\mathsf{otm}^i_{2,t,\alpha,\beta}$ on behalf of every honest party P_i such that $i \in H \setminus \{i^*\}$ for all choices of $\alpha, \beta \in \{0,1\}$. More specifically, S only generates one of these four values; namely, $\mathsf{otm}^i_{2,t,\alpha^*,\beta^*}$ which is now generated as $\mathsf{OT}_2(\mathsf{otm}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t})$ instead of $\mathsf{OT}_2(\mathsf{otm}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$.

  Indistinguishability between hybrids $\mathsf{Hyb}_{t,2}$ and $\mathsf{Hyb}_{t,1}$ follows directly from the sender's security of underlying semi-honest oblivious transfer protocol.

- $\mathsf{Hyb}_{t,3}$: Skip this hybrid, if $i^* \notin H$. This hybrid is same as $\mathsf{Hyb}_{t,2}$ except that we change how S generates the Round-1 message on behalf of P_{i^*}. Specifically, the simulator S generates $\mathsf{otm}_{1,t,\alpha,\beta}$ as is done in the Hyb_t. In a bit more detail, for all $\alpha, \beta \in \{0,1\}$, S generates $\mathsf{otm}_{1,t,\alpha,\beta}$ as $\mathsf{OT}_1(1^\lambda, \mathsf{Z}_t; \omega_{t,\alpha,\beta})$ rather than $\mathsf{OT}_1(1^\lambda, v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta); \omega_{t,\alpha,\beta})$.

  Indistinguishability between hybrids $\mathsf{Hyb}_{t,2}$ and $\mathsf{Hyb}_{t,3}$ follows directly by a sequence of 3 sub-hybrids each one relying on the receiver's security of underlying semi-honest oblivious transfer protocol. Observe here that the security reduction crucially relies on the fact that $\widetilde{\mathsf{P}}^{i,t}$ only contains $\omega_{t,\alpha^*,\beta^*}$ (i.e., does not have $\omega_{t,\alpha,\beta}$ for $\alpha \neq \alpha^*$ or $\beta \neq \beta^*$).

  Finally, observe that $\mathsf{Hyb}_{t,3}$ is the same as hybrid Hyb_t.

■

3.5 Two-Round MPC: Malicious Case

In this section, we give our construction of two-round multiparty computation protocol in the malicious case with security against static corruptions based on any two-round malicious oblivious transfer protocol with equivocal receiver security (which from Lemma 3.2.2 follows from any two-round malicious oblivious transfer). This is achieved by designing a compiler that takes any conforming arbitrary (polynomial) round MPC protocol Φ with malicious security and squishes it to two rounds.

3.5.1 Our Compiler

We give our construction of two-round MPC in Figure 3.5 and the circuit that needs to be garbled (repeatedly) is shown in Figure 3.4 (same as the semi-honest case). Our compiler is essentially the same as the semi-honest case and uses an n-party, conforming malicious MPC protocol Φ, a garbling scheme for circuits $(\mathsf{Garb}, \mathsf{Eval})$ and a malicious two-round oblivious transfer protocol with equivocal receiver security $(K_{\mathsf{OT}}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$.

Another technical issue arises because the adversary may wait to receiver first round messages that S sends on the behalf of honest parties before the corrupted parties send out their first round messages. Recall that by sending the receiver OT messages in the first round, every party "commits" to all its future messages that it will send in the computation phase of the protocol. Thus, the ideal world simulator S must somehow commit to the messages generated on behalf of the honest party before extracting the adversary's effective input. To get around this issue, we use the equivocability property of the OT using which the simulator can equivocate its first round messages after learning the malicious adversary's effective input.

Theorem 3.5.1 *Let Φ be a polynomial round, n-party malicious MPC protocol computing a function $f : (\{0,1\}^m)^n \to \{0,1\}^*$, $(\mathsf{Garb}, \mathsf{Eval})$ be a garbling scheme for circuits, and $(K_{\mathsf{OT}}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ be a maliciously secure (with equivocal receiver security) two-round OT protocol. The protocol described in Figure 3.5 is a two-round, n-party malicious MPC protocol computing f against static corruptions.*

We prove this theorem in the rest of the section. As in the semi-honest case, it is useful to keep in mind that for every $i, j \in [n]$ and $k \in [\ell]$, we have that $\mathsf{st}_{i,k} \oplus v_{i,k} = \mathsf{st}_{j,k} \oplus v_{j,k}$. Let us denote this shared value by st^*. Also, we denote the transcript of the interaction in the computation phase by $\mathsf{Z} \in \{0,1\}^t$.

3.5.2 Simulator

Let $\mathcal{A}$ be a malicious adversary corrupting a subset of parties and let $H \subseteq [n]$ be the set of honest/uncorrupted parties. Since we assume that the adversary is static, this set is fixed before the execution of the protocol. Below we provide the notion of faithful execution and then describe our simulator.

Faithful Execution. In the first round of our compiled protocol, $\mathcal{A}$ provides z_i for every $i \in [n] \setminus H$ and $\mathsf{otm}_{1,t,\alpha,\beta}$ for every $t \in \cup_{i \in [n] \setminus h}$ and $\alpha, \beta \in \{0,1\}$. These values act as "binding" commitments to all of the adversary's future choices. All these committed choices can be extracted using the extractor Ext_2. Let $b_{t,\alpha,\beta}$ be the value extracted from $\mathsf{otm}_{1,t,\alpha,\beta}$. Intuitively speaking, a faithful execution is an execution that is consistent with these extracted values.

Common Random/Reference String: For each $t \in T, \alpha, \beta \in \{0,1\}$ sample $\sigma_{t,\alpha,\beta} \leftarrow K_{\mathsf{OT}}(1^\lambda)$ and output $\{\sigma_{t,\alpha,\beta}\}_{t\in[T],\alpha,\beta\in\{0,1\}}$ as the common random/reference string.

Round-1: Each party P_i does the following:

1. Compute $(z_i, v_i) \leftarrow \mathsf{pre}(1^\lambda, i, x_i)$.
2. For each t such that $\phi_t = (i, f, g, h)$ (A_i is the set of such values of t), for each $\alpha, \beta \in \{0,1\}$
$$\mathsf{otm}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}_1(\sigma_{t,\alpha,\beta}, v_{i,h} \oplus \mathsf{NAND} v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta; \omega_{t,\alpha,\beta}).$$
3. Send $\left(z_i, \{\mathsf{otm}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}\right)$ to every other party.

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

1. Set $\mathsf{st}_i := (z_1 \| \dots \| z_{i-1} \| z_i \| z_{i+1} \| \dots \| z_n) \oplus v_i$.
2. Set $\overline{\mathsf{lab}}^{i,T+1} := \{\mathsf{lab}^{i,T+1}_{k,0}, \mathsf{lab}^{i,T+1}_{k,1}\}_{k\in[\ell]}$ where for each $k \in [\ell]$ and $b \in \{0,1\}$ $\mathsf{lab}^{i,T+1}_{k,b} := 0^\lambda$.
3. **for** each t from T down to 1,
   a) Parse ϕ_t as (i^*, f, g, h).
   b) If $i = i^*$ then compute (where P is described in Figure 3.4) $\left(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{lab}}^{i,t}\right) \leftarrow \mathsf{Garb}(1^\lambda, \mathsf{P}[i, \phi_t, v_i, \{\omega_{t,\alpha,\beta}\}_{\alpha,\beta}, \bot, \overline{\mathsf{lab}}^{i,t+1}])$.
   c) If $i \neq i^*$ then for every $\alpha, \beta \in \{0,1\}$, set $\mathsf{otm}^i_{2,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\sigma_{t,\alpha,\beta}, \mathsf{otm}_{1,t,\alpha,\beta}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$ and compute $\left(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{lab}}^{i,t}\right) \leftarrow \mathsf{Garb}(1^\lambda, \mathsf{P}[i, \phi_t, v_i, \bot, \{\mathsf{otm}^i_{2,t,\alpha,\beta}\}_{\alpha,\beta}, \overline{\mathsf{lab}}^{i,t+1}])$.
4. Send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \{\mathsf{lab}^{i,1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]}\right)$ to every other party.

Evaluation: To compute the output of the protocol, each party P_i does the following:

1. For each $j \in [n]$, let $\widetilde{\mathsf{lab}}^{j,1} := \{\mathsf{lab}^{j,1}_k\}_{k\in[\ell]}$ be the labels received from party P_j at the end of round 2.
2. **for** each t from 1 to T do:
   a) Parse ϕ_t as (i^*, f, g, h).
   b) Compute $((\alpha, \beta, \gamma), \omega, \widetilde{\mathsf{lab}}^{i^*,t+1}) := \mathsf{Eval}(\widetilde{\mathsf{P}}^{i^*,t}, \widetilde{\mathsf{lab}}^{i^*,t})$.
   c) Set $\mathsf{st}_{i,h} := \gamma \oplus v_{i,h}$.
   d) **for** each $j \neq i^*$ do:
      i. Compute $(\mathsf{otm}_2, \{\mathsf{lab}^{j,t+1}_k\}_{k\in[\ell]\setminus\{h\}}) := \mathsf{Eval}(\widetilde{\mathsf{P}}^{j,t}, \widetilde{\mathsf{lab}}^{j,t})$.
      ii. Recover $\mathsf{lab}^{j,t+1}_h := \mathsf{OT}_3(\sigma_{t,\alpha,\beta}, \mathsf{otm}_2, \omega)$.
      iii. Set $\widetilde{\mathsf{lab}}^{j,t+1} := \{\mathsf{lab}^{j,t+1}_k\}_{k\in[\ell]}$.
3. Compute the output as $\mathsf{post}(\mathsf{st}_i)$.

Figure 3.5: Two-round Malicious MPC.

More formally, we define an interactive procedure $\mathsf{Faithful}(i, \{z_i\}_{i\in[n]}, \{b_{t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta})$ that on input $i \in [n]$, $\{z_i\}_{i\in[n]}$, $\{b_{t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}$ produces protocol Φ message on behalf of party P_i (acting consistently/faithfully with the extracted values) as follows:

1. Set $\mathsf{st}^* := z_1 \| \ldots \| z_n$.
2. For $t \in \{1 \cdots T\}$
   a) Parse $\phi_t = (i^*, f, g, h)$.
   b) If $i \neq i^*$ then it waits for a bit from P_{i^*} and sets st^*_h to be the received bit once it is received.
   c) Set $\mathsf{st}^*_h := b_{t,\mathsf{st}^*_f,\mathsf{st}^*_g}$ and output it to all the other parties.

We will later argue that any deviation from the faithful execution by the adversary $\mathcal{A}$ on behalf of the corrupted parties (during the second round of our compiled protocol) will be detected. Additionally, we prove that such deviations do not hurt the security of the honest parties.

Description of the Simulator. We give the description of the ideal world adversary S that simulates the view of the real world adversary $\mathcal{A}$. S will internally use the malicious simulator Sim_Φ for Φ, the extractor $\mathsf{Ext} = (\mathsf{Ext}_1, \mathsf{Ext}_2)$ implied by the sender security of two-round OT, the simulator Sim_{Eq} implied by the equivocal receiver's security and the simulator Sim_G for garbling scheme for circuits. Recall that $\mathcal{A}$ is static and hence the set of honest parties H is known before the execution of the protocol.

Simulating the interaction with $\mathcal{Z}$. For every input value for the set of corrupted parties that S receives from $\mathcal{Z}$, S writes that value to $\mathcal{A}$'s input tape. Similarly, the output of $\mathcal{A}$ is written as the output on S's output tape.

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

- **Generation of the common random/reference string**: S generates the common random/reference string as follows:
  1. For each $i \in H, t \in A_i, \alpha, \beta \in \{0,1\}$ set $(\sigma_{t,\alpha,\beta}, (\mathsf{otm}_{1,t,\alpha,\beta}, \omega^0_{t,\alpha,\beta}, \omega^1_{t,\alpha,\beta})) \leftarrow \mathsf{Sim}_{Eq}(1^\lambda)$ (using equivocal simulator).
  2. For each $i \in [n] \setminus H, \alpha, \beta \in \{0,1\}$ and $t \in A_i$ generate $(\sigma_{t,\alpha,\beta}, \tau_{t,\alpha,\beta}) \leftarrow \mathsf{Ext}_1(1^\lambda)$ (using the extractor of the OT protocol).
  3. Output the common random/reference string as $\{\sigma_{t,\alpha,\beta}\}_{t,\alpha,\beta}$.

- **Initialization**: S executes the simulator (against malicious adversary's) $\mathsf{Sim}_\Phi(1^\lambda)$ to obtain $\{z_i\}_{i\in H}$. Moreover, S starts the real-world adversary $\mathcal{A}$. We next describe how S provides its messages to Sim_Φ and $\mathcal{A}$.

- **Round-1 messages from S to $\mathcal{A}$:** For each $i \in H$, S sends $(z_i, \{\mathsf{otm}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}})$ to the adversary $\mathcal{A}$ on behalf of the honest party P_i.

- **Round-1 messages from $\mathcal{A}$ to S:** Corresponding to every $i \in [n]\setminus H$, S receives from the adversary $\mathcal{A}$ the value $(z_i, \{\mathsf{otm}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}})$ on behalf of the corrupted party P_i. Next, for each $i \in [n]\setminus H, t \in A_i, \alpha, \beta \in \{0,1\}$ extract $b_{t,\alpha,\beta} := \mathsf{Ext}_2(\tau_{t,\alpha,\beta}, \mathsf{otm}_{1,t,\alpha,\beta})$.

- **Completing the execution with the Sim_Φ**: For each $i \in [n] \setminus H$, S sends z_i to Sim_Φ on behalf of the corrupted party P_i. This starts the computation phase of Φ with the simulator Sim_Φ. S provides computation phase messages to Sim_Φ by following a faithful execution. More formally, for every corrupted party P_i where $i \in [n] \setminus H$, S generates messages on behalf of P_i for Sim_Φ using the procedure $\mathsf{Faithful}(i, \{z_i\}_{i\in[n]}, \{b_{t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta})$. At some point during the execution, Sim_Φ will return the extracted inputs $\{x_i\}_{i\in[n]\setminus H}$ of the corrupted parties. For each $i \in [n] \setminus H$, S sends $(\mathsf{input}, \mathsf{sid}, \{P_1 \cdots P_n\}, P_i, x_i)$ to the ideal functionality implementing f and obtains the output y which is provided to Sim_Φ. Finally, at some point the faithful execution completes.

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

- **Round-2 messages from S to $\mathcal{A}$:**

  For each $i \in H$, the simulator S generates the second round message on behalf of party P_i as follows:

  1. For each $k \in [\ell]$ set $\mathsf{lab}^{i,T+1}_k := 0^\lambda$.
  2. **for** each t from T down to 1,
     a) Parse ϕ_t as (i^*, f, g, h).
     b) Set $\alpha^* := \mathsf{st}^*_f$, $\beta^* := \mathsf{st}^*_g$, and $\gamma^* := \mathsf{st}^*_h$.
     c) If $i = i^*$ then compute

$$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_\mathsf{G}\left(1^\lambda, 1^{|\mathsf{P}|}, 1^\ell, \left((\alpha^*, \beta^*, \gamma^*), \omega_{t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_k\}_{k\in[\ell]}\right)\right).$$

     d) If $i \neq i^*$ then set $\mathsf{otm}^i_{2,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2(\sigma_{t,\alpha^*,\beta^*}, \mathsf{otm}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_h, \mathsf{lab}^{i,t+1}_h)$ and compute

$$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_\mathsf{G}\left(1^\lambda, 1^{|\mathsf{P}|}, 1^\ell, \left(\mathsf{otm}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_k\}_{k\in[\ell]\setminus\{h\}}\right)\right).$$

3. Send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \{\mathsf{lab}_k^{i,1}\}_{k\in[\ell]}\right)$ to every other party.

- **Round-2 messages from $\mathcal{A}$ to S:** For every $i \in [n] \setminus H$, S obtains the second round message from $\mathcal{A}$ on behalf of the malicious parties. Subsequent to obtaining these messages, S executes the garbled circuits provided by $\mathcal{A}$ on behalf of the corrupted parties to see the execution of garbled circuits proceeds consistently with the expected faithful execution. If the computation succeeds then, S sends $(\mathsf{generateOutput}, \mathsf{sid}, \{P_1 \cdots P_n\})$ to the ideal functionality. Otherwise, it sends $(\mathsf{abort}, \mathsf{sid}, \{P_1 \cdots P_n\})$.

3.5.3 Proof of Indistinguishability

We now show that no environment $\mathcal{Z}$ can distinguish whether it is interacting with a real world adversary $\mathcal{A}$ or an ideal world adversary S. We prove this via an hybrid argument with $T + 3$ hybrids.

- $\mathcal{H}_{Real}$: This hybrid is the same as the real world execution.
- $\mathcal{H}_0$: In this hybrid we start by changing the distribution of the common random string. Specifically, the common random string for the corrupted parties is generated as is done in the simulation. More formally, S generates the common random/reference string as follows:

  1. For each $i \in [n] \setminus H, \alpha, \beta \in \{0,1\}$ and $t \in A_i$ generate $(\sigma_{t,\alpha,\beta}, \tau_{t,\alpha,\beta}) \leftarrow \mathsf{Ext}_1(1^\lambda)$ (using the extractor of the OT protocol).

     Corresponding to every $i \in [n] \setminus H$, $\mathcal{A}$ sends $(z_i, \{\mathsf{otm}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}})$ on behalf of the corrupted party P_i as its first round message. For each $i \in [n] \setminus H, t \in A_i, \alpha, \beta \in \{0,1\}$ in this hybrid we extract $b_{t,\alpha,\beta} := \mathsf{Ext}_2(\tau_{t,\alpha,\beta}, \mathsf{otm}_{1,t,\alpha,\beta})$.

  The indistinguishability between hybrids $\mathcal{H}_{Real}$ and $\mathcal{H}_0$ follow from a reduction to the sender's security of the two-round OT protocol.

- $\mathcal{H}_t$ (where $t \in \{0, \ldots T\}$): Hybrid $\mathcal{H}_t$ (for $t \in \{1 \cdots T\}$) is the same as hybrid $\mathcal{H}_{t-1}$ except we change the distribution of the OT messages (both from the first and the second round of the protocol) and the garbled circuits (from the second round) that play a role in the execution of the t^{th} round of the protocol Φ; namely, the action $\phi_t = (i^*, f, g, h)$. We describe the changes more formally below.

  For each $i \in [n] \setminus H$, in this hybrid S (in his head) completes an execution of Φ using honest party inputs and randomness. In this execution, the messages on behalf of corrupted parties are generated via faithful execution. Specifically, S sends $\{z_i\}_{i\in[n]\setminus H}$ to the honest parties on behalf of the corrupted party P_i in this mental execution of Φ. This starts the computation phase of Φ. In this computation phase, S generates honest party messages using the inputs and random coins of the honest parties and generates the messages of the each malicious party P_i by executing $\mathsf{Faithful}\left(i, \{z_i\}_{i\in[n]\setminus H}, \{b_{t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta}\right)$.

Let $\mathsf{Z} \in \{0,1\}^T$ be the transcript, st^* be the local state of one of the corrupted party the end of faithful execution and let st_i^{t+1} be the state of honest party $i \in H$ at the end of the t-th round of the computation phase. Finally, let $\alpha^* := \mathsf{st}_f^*$, $\beta^* := \mathsf{st}_g^*$ and $\gamma^* := \mathsf{st}_h^*$. In hybrid $\mathcal{H}_t$ we make the following changes with respect to hybrid $\mathcal{H}_{t-1}$:

- If $i^* \notin H$ then skip these changes. S makes two changes in how it generates messages on behalf of P_{i^*}. First, for all $\alpha, \beta \in \{0,1\}$, S computes $(\sigma_{t,\alpha,\beta}, (\mathsf{otm}_{1,t,\alpha,\beta}, \omega^0_{t,\alpha,\beta}, \omega^1_{t,\alpha,\beta})) \leftarrow \mathsf{Sim}_{Eq}(1^\lambda)$ (using equivocal simulator) and sets $\omega_{t,\alpha^*,\beta^*}$ as $\omega^{\mathsf{Z}_t}_{t,\alpha^*,\beta^*}$ rather than $\omega^{v_{i,h} \oplus \mathsf{NAND} v_{i,f} \oplus \alpha^*, v_{i,g} \oplus \beta^*}_{t,\alpha^*,\beta^*}$ (note that these two values are the same when using the honest party's input and randomness). Second, it generates the garbled circuit

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

  where $\{\mathsf{lab}^{i^*,t+1}_{k,\mathsf{st}^{t+1}_{i,k}}\}_{k\in[\ell]}$ are the honestly generates input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i^*,t+1}$.

- S makes the following two changes in how it generates messages for other honest parties P_i (i.e., $i \in H \setminus \{i^*\}$). S does not generate four $\mathsf{otm}^i_{2,t,\alpha,\beta}$ values but just one of them; namely, S generates $\mathsf{otm}^i_{2,t,\alpha^*,\beta^*}$ as $\mathsf{OT}_2(\sigma_{t,\alpha^*,\beta^*}, \mathsf{otm}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t})$ rather than $\mathsf{OT}_2(\sigma_{t,\alpha^*,\beta^*}, \mathsf{otm}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$. Second it generates the garbled circuit

$$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}_k^{i,t}\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^\lambda, 1^{|\mathsf{P}|}, 1^\ell, \left(\mathsf{otm}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^{t+1}_{i,k}}\}_{k\in[\ell]\setminus\{h\}}\right)\right),$$

  where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}^{t+1}_{i,k}}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i,t+1}$.

  Indistinguishability between $\mathcal{H}_{t-1}$ and $\mathcal{H}_t$ is proved in Lemma 3.5.2.

- $\mathcal{H}_{T+1}$: In this hybrid, we modify the output phase of the computation to execute the garbled circuits provided by $\mathcal{A}$ on behalf of the corrupted parties and see if the execution of garbled circuits proceeds consistently with the transcript Z. If the computation succeeds then for each $i \in H$, we instruct the parties in H to output y (which is the output obtained by all parties in the execution of Φ); else, we instruct them to output $\perp$. This hybrid is computationally close to $\mathcal{H}_T$ from the authenticity property of the input labels.

- $\mathcal{H}_{T+2}$: In this hybrid we just change how the transcript Z, $\{z_i\}_{i\in H}$, random coins of malicious parties and value st^* are generated. Instead of generating these using honest party inputs in execution with a faithful execution of Φ, we generate it via the simulator Sim_Φ (of the maliciously secure protocol Φ). In other words, we execute the

simulator Sim_Φ where messages on behalf of each corrupted party P_i are generated using $\mathsf{Faithful}(i, \{z_i\}_{i\in[n]\setminus H}, \{b_{t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta})$. (Note that Sim_Φ might rewind $\mathsf{Faithful}$. This can be achieved since $\mathsf{Faithful}$ is just a polynomial time interactive procedure that can also be rewound.).

The indistinguishability between hybrids $\mathcal{H}_{T+1}$ and $\mathcal{H}_{T+2}$ follows directly from the malicious security of the protocol Φ. Finally note that $\mathcal{H}_{T+1}$ is same as the ideal execution (i.e., the simulator described in the previous subsection).

Lemma 3.5.2 *Assuming malicious, equivocal receiver security of the two-round OT protocol and the security of the garbling scheme, for all $t \in \{1 \dots T\}$ hybrids $\mathcal{H}_{t-1}$ and $\mathcal{H}_t$ are computationally indistinguishable.*

Proof Using the same notation as before, let $\phi_t = (i^*, f, g, h)$, st_i^{t+1} be the state of P_i at the end of round t for each $i \in H$, Z be the transcript and st^* be the local state of one of the corrupted party the end of faithful execution, and $\alpha^* := \mathsf{st}_f^*$, $\beta^* := \mathsf{st}_g^*$ and $\gamma^* := \mathsf{st}_h^*$. The indistinguishability between hybrids $\mathcal{H}_{t-1}$ and $\mathcal{H}_t$ follows by a sequence of three sub-hybrids $\mathcal{H}_{t,1}$, $\mathcal{H}_{t,2}$, and $\mathcal{H}_{t,3}$.

- $\mathcal{H}_{t,1}$: Hybrid $\mathcal{H}_{t,1}$ is same as hybrid $\mathcal{H}_{t-1}$ except that S now generates the garbled circuits $\widetilde{\mathsf{P}}^{i,t}$ for each $i \in H$ in a simulated manner (rather than generating them honestly). Specifically, instead of generating each garbled circuit and input labels $\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}_k^{i,t}\}_{k\in[\ell]}\right)$ honestly, they are generated via the simulator by hard coding the output of the circuit itself. In a bit more details, parse ϕ_t as (i^*, f, g, h).
  - If $i = i^*$ then

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

    where $\{\mathsf{lab}_{k,\mathsf{st}_{i,k}^{t+1}}^{i,t+1}\}_{k\in[\ell]}$ are the honestly generates input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i,t+1}$.
  - If $i \neq i^*$ then

$$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}_k^{i,t}\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^\lambda, 1^{|\mathsf{P}|}, 1^\ell, \left(\mathsf{otm}_{2,t,\alpha^*,\beta^*}^i, \{\mathsf{lab}_{k,\mathsf{st}_{i,k}^{t+1}}^{i,t+1}\}_{k\in[\ell]\setminus\{h\}}\right)\right),$$

    where $\{\mathsf{lab}_{k,\mathsf{st}_{i,k}^{t+1}}^{i,t+1}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i,t+1}$.

  The indistinguishability between hybrids $\mathcal{H}_{t,1}$ and $\mathcal{H}_{t-1}$ follows by $|H|$ invocations of security of the garbling scheme.

- $\mathcal{H}_{t,2}$: Skip this hybrid if there does not exist $i \neq i^*$ such that $i \in H$. In this hybrid, we change how S generates the $\mathsf{otm}^i_{2,t,\alpha,\beta}$ on behalf of every honest party P_i such that $i \in H \setminus \{i^*\}$ for all choices of $\alpha, \beta \in \{0,1\}$. More specifically, S only generates one of these four values; namely, $\mathsf{otm}^i_{2,t,\alpha^*,\beta^*}$ which is now generated as $\mathsf{OT}_2(\sigma_{t,\alpha^*,\beta^*}, \mathsf{otm}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t})$ instead of $\mathsf{OT}_2(\sigma_{t,\alpha^*,\beta^*}, \mathsf{otm}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$.

  Indistinguishability between hybrids $\mathcal{H}_{t,2}$ and $\mathcal{H}_{t,1}$ follows directly from the sender's security of underlying malicious oblivious transfer protocol.

- $\mathcal{H}_{t,3}$: Skip this hybrid, if $i^* \notin H$. This hybrid is same as $\mathcal{H}_{t,2}$ except that we change how S generates the Round-1 message on behalf of P_{i^*}. Specifically, the simulator S generates $\mathsf{otm}_{1,t,\alpha,\beta}$ as is done in the $\mathcal{H}_t$. In a bit more detail, S computes $(\sigma_{t,\alpha,\beta}, (\mathsf{otm}_{1,t,\alpha,\beta}, \omega^0_{t,\alpha,\beta}, \omega^1_{t,\alpha,\beta})) \leftarrow \mathsf{Sim}_{Eq}(1^\lambda)$ (using equivocal simulator) and sets $\omega_{t,\alpha^*,\beta^*}$ as $\omega^{\mathsf{Z}_t}_{t,\alpha^*,\beta^*}$ rather than $\omega^{v_{i,h} \oplus \mathsf{NAND} v_{i,f} \oplus \alpha^*, v_{i,g} \oplus \beta^*}_{t,\alpha^*,\beta^*}$ (note that these two values are the same when using the honest party's input and randomness).

  We now argue indistingiuishability between $\mathcal{H}_{t,2}$ and $\mathcal{H}_{t,3}$ using the equivocal receiver security. We interact with the equivocal security challenger four times. For each α, β, we obtain $\sigma_{t,\alpha,\beta}, \mathsf{otm}_{1,t,\alpha,\beta}$ and $\omega^{v_{i,h} \oplus \mathsf{NAND} v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta}_{t,\alpha,\beta}$. We use this to generate the first round message and the second round messages of the protocol. Note that if these values were generated honestly then the distribution produced is identical to $\mathcal{H}_{t,2}$. Else, it is distributed identically to $\mathcal{H}_{t,3}$. Finally, observe that $\mathcal{H}_{t,3}$ is the same as hybrid $\mathcal{H}_t$.

■

Bibliography

[ABT18] Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. "Perfect Secure Computation in Two Rounds". In: *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I*. Ed. by Amos Beimel and Stefan Dziembowski. Vol. 11239. Lecture Notes in Computer Science. Springer, 2018, pp. 152–174. DOI: 10.1007/978-3-030-03807-6_6. URL: https://doi.org/10.1007/978-3-030-03807-6%5C_6.

[ABT19] Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. "Degree 2 is Complete for the Round-Complexity of Malicious MPC". In: *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. Lecture Notes in Computer Science. Springer, 2019, pp. 504–531. DOI: 10.1007/978-3-030-17656-3_18. URL: https://doi.org/10.1007/978-3-030-17656-3%5C_18.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. "Cryptography in NC^0". In: *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*. IEEE Computer Society, 2004, pp. 166–175. DOI: 10.1109/FOCS.2004.20. URL: https://doi.org/10.1109/FOCS.2004.20.

[AIK05] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. "Computationally Private Randomizing Polynomials and Their Applications". In: *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*. IEEE Computer Society, 2005, pp. 260–274. DOI: 10.1109/CCC.2005.9. URL: https://doi.org/10.1109/CCC.2005.9.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. "Priced Oblivious Transfer: How to Sell Digital Goods". In: *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*. Ed. by Birgit Pfitzmann. Vol. 2045. Lecture Notes in Computer Science. Springer, 2001, pp. 119–135. DOI: 10.1007/3-540-44987-6_8. URL: https://doi.org/10.1007/3-540-44987-6%5C_8.

[Ana+18] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. "Round-Optimal Secure Multiparty Computation with Honest Majority". In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10992. Lecture Notes in Computer Science. Springer, 2018, pp. 395–424. DOI: 10.1007/978-3-319-96881-0_14. URL: https://doi.org/10.1007/978-3-319-96881-0%5C_14.

[Ana+19] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. "Two Round Information-Theoretic MPC with Malicious Security". In: *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. Lecture Notes in Computer Science. Springer, 2019, pp. 532–561. DOI: 10.1007/978-3-030-17656-3_19. URL: https://doi.org/10.1007/978-3-030-17656-3%5C_19.

[App+20] Benny Applebaum, Zvika Brakerski, Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. "Separating Two-Round Secure Computation From Oblivious Transfer". In: *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*. Ed. by Thomas Vidick. Vol. 151. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 71:1–71:18. DOI: 10.4230/LIPIcs.ITCS.2020.71. URL: https://doi.org/10.4230/LIPIcs.ITCS.2020.71.

[Ash+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. "Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE". In: *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. Lecture Notes in Computer Science. Springer, 2012, pp. 483–501. DOI: 10.1007/978-3-642-29011-4_29. URL: https://doi.org/10.1007/978-3-642-29011-4%5C_29.

[Bad+18] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. "Promise Zero Knowledge and Its Applications to Round Optimal MPC". In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10992. Lecture Notes in Computer Science. Springer, 2018, pp. 459–487. DOI: 10.1007/978-3-319-96881-0_16. URL: https://doi.org/10.1007/978-3-319-96881-0%5C_16.

[Bar+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. "On the (Im)possibility of Obfuscating Programs". In: *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 1–18. DOI: 10.1007/3-540-44647-8_1. URL: https://doi.org/10.1007/3-540-44647-8%5C_1.

[Bar+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. "Secure Computation Without Authentication". In: *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, 2005, pp. 361–377. DOI: 10.1007/11535218_22. URL: https://doi.org/10.1007/11535218%5C_22.

[Bar+20] James Bartusek, Sanjam Garg, Daniel Masny, and Pratyay Muhkerjee. "Reusable Two-Round MPC from DDH". In: *IACR Cryptology ePrint Archive* 2020 (2020), p. 170. URL: https://eprint.iacr.org/2020/170.

[Ben+18] Fabrice Benhamouda, Huijia Lin, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. "Two-Round Adaptively Secure Multiparty Computation from Standard Assumptions". In: *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I*. Ed. by Amos Beimel and Stefan Dziembowski. Vol. 11239. Lecture Notes in Computer Science. Springer, 2018, pp. 175–205. DOI: 10.1007/978-3-030-03807-6_7. URL: https://doi.org/10.1007/978-3-030-03807-6%5C_7.

[BF01] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 213–229. DOI: 10.1007/3-540-44647-8_13. URL: https://doi.org/10.1007/3-540-44647-8%5C_13.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Breaking the Circuit Size Barrier for Secure Computation Under DDH". In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, 2016, pp. 509–539. DOI: 10.1007/978-3-662-53018-4_19. URL: https://doi.org/10.1007/978-3-662-53018-4%5C_19.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Group-Based Secure Computation: Optimizing Rounds, Communication, and Computation". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on*

the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10211. Lecture Notes in Computer Science. 2017, pp. 163–193. DOI: 10.1007/978-3-319-56614-6_6. URL: https://doi.org/10.1007/978-3-319-56614-6%5C_6.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. Ed. by Janos Simon. ACM, 1988, pp. 1–10. DOI: 10.1145/62212.62213. URL: https://doi.org/10.1145/62212.62213.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. "Foundations of garbled circuits". In: *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*. Ed. by Ting Yu, George Danezis, and Virgil D. Gligor. ACM, 2012, pp. 784–796. DOI: 10.1145/2382196.2382279. URL: https://doi.org/10.1145/2382196.2382279.

[BL18] Fabrice Benhamouda and Huijia Lin. "k-Round Multiparty Computation from k-Round Oblivious Transfer via Garbled Interactive Circuits". In: *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 500–532. DOI: 10.1007/978-3-319-78375-8_17. URL: https://doi.org/10.1007/978-3-319-78375-8%5C_17.

[BM09] Boaz Barak and Mohammad Mahmoody-Ghidary. "Merkle Puzzles Are Optimal - An $O(n^2)$-Query Attack on Any Key Exchange from a Random Oracle". In: *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*. Ed. by Shai Halevi. Vol. 5677. Lecture Notes in Computer Science. Springer, 2009, pp. 374–390. DOI: 10.1007/978-3-642-03356-8_22. URL: https://doi.org/10.1007/978-3-642-03356-8%5C_22.

[BM89] Mihir Bellare and Silvio Micali. "Non-Interactive Oblivious Transfer and Spplications". In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 547–557. DOI: 10.1007/0-387-34805-0_48. URL: https://doi.org/10.1007/0-387-34805-0%5C_48.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. "The Round Complexity of Secure Protocols (Extended Abstract)". In: *Proceedings of the 22nd Annual*

ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA. Ed. by Harriet Ortiz. ACM, 1990, pp. 503–513. DOI: 10.1145/100216.100287. URL: https://doi.org/10.1145/100216.100287.

[Boy+18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. "Foundations of Homomorphic Secret Sharing". In: *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*. Ed. by Anna R. Karlin. Vol. 94. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 21:1–21:21. DOI: 10.4230/LIPIcs.ITCS.2018.21. URL: https://doi.org/10.4230/LIPIcs.ITCS.2018.21.

[BP16] Zvika Brakerski and Renen Perlman. "Lattice-Based Fully Dynamic Multi-key FHE with Short Ciphertexts". In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, 2016, pp. 190–213. DOI: 10.1007/978-3-662-53018-4_8. URL: https://doi.org/10.1007/978-3-662-53018-4%5C_8.

[Can+02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. "Universally composable two-party and multi-party secure computation". In: *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*. Ed. by John H. Reif. ACM, 2002, pp. 494–503. DOI: 10.1145/509907.509980. URL: https://doi.org/10.1145/509907.509980.

[Can00] Ran Canetti. "Security and Composition of Multiparty Cryptographic Protocols". In: *J. Cryptology* 13.1 (2000), pp. 143–202. DOI: 10.1007/s001459910006. URL: https://doi.org/10.1007/s001459910006.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888. URL: https://doi.org/10.1109/SFCS.2001.959888.

[Can04] Ran Canetti. "Universally Composable Signature, Certification, and Authentication". In: *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*. IEEE Computer Society, 2004, p. 219. DOI: 10.1109/CSFW.2004.24. URL: http://doi.ieeecomputersociety.org/10.1109/CSFW.2004.24.

[CF01] Ran Canetti and Marc Fischlin. "Universally Composable Commitments". In: *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 19–40. DOI: 10.1007/3-540-44647-8_2. URL: https://doi.org/10.1007/3-540-44647-8%5C_2.

[CGZ20] Ran Cohen, Juan A. Garay, and Vassilis Zikas. "Broadcast-Optimal Two-Round MPC". In: *To appear in EUROCRYPT* 2020 (2020), p. 1183. URL: https://eprint.iacr.org/2019/1183.

[Cho+19] Arka Rai Choudhuri, Michele Ciampi, Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. "On Round Optimal Secure Multiparty Computation from Minimal Assumptions". In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 216. URL: https://eprint.iacr.org/2019/216.

[CK91] Benny Chor and Eyal Kushilevitz. "A Zero-One Law for Boolean Privacy". In: *SIAM J. Discrete Math.* 4.1 (1991), pp. 36–47. DOI: 10.1137/0404004. URL: https://doi.org/10.1137/0404004.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. "Adaptive Hardness and Composable Security in the Plain Model from Standard Assumptions". In: *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*. IEEE Computer Society, 2010, pp. 541–550. DOI: 10.1109/FOCS.2010.86. URL: https://doi.org/10.1109/FOCS.2010.86.

[CR03] Ran Canetti and Tal Rabin. "Universal Composition with Joint State". In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 265–281. DOI: 10.1007/978-3-540-45146-4_16. URL: https://doi.org/10.1007/978-3-540-45146-4%5C_16.

[Döt+20] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. *Two-Round Oblivious Transfer from CDH or LPN*. To appear in EUROCRYPT. https://eprint.iacr.org/2019/414. 2020.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. "A Randomized Protocol for Signing Contracts". In: *Commun. ACM* 28.6 (1985), pp. 637–647. DOI: 10.1145/3812.3818. URL: http://doi.acm.org/10.1145/3812.3818.

[Gar+13a] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. "Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits". In: *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. IEEE Computer Society, 2013, pp. 40–49. DOI: 10.1109/FOCS.2013.13. URL: https://doi.org/10.1109/FOCS.2013.13.

[Gar+13b] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. "Witness encryption and its applications". In: *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM, 2013, pp. 467–476. DOI: 10.1145/2488608.2488667. URL: https://doi.org/10.1145/2488608.2488667.

[Gar+14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. "Two-Round Secure MPC from Indistinguishability Obfuscation". In: *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*. Ed. by Yehuda Lindell. Vol. 8349. Lecture Notes in Computer Science. Springer, 2014, pp. 74–94. DOI: 10.1007/978-3-642-54242-8_4. URL: https://doi.org/10.1007/978-3-642-54242-8%5C_4.

[GIS18] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. "Two-Round MPC: Information-Theoretic and Black-Box". In: *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I*. Ed. by Amos Beimel and Stefan Dziembowski. Vol. 11239. Lecture Notes in Computer Science. Springer, 2018, pp. 123–151. DOI: 10.1007/978-3-030-03807-6_5. URL: https://doi.org/10.1007/978-3-030-03807-6%5C_5.

[GLS15] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. "Constant-Round MPC with Fairness and Guarantee of Output Delivery". In: *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*. Ed. by Rosario Gennaro and Matthew Robshaw. Vol. 9216. Lecture Notes in Computer Science. Springer, 2015, pp. 63–82. DOI: 10.1007/978-3-662-48000-7_4. URL: https://doi.org/10.1007/978-3-662-48000-7%5C_4.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks". In: *SIAM J. Comput.* 17.2 (1988), pp. 281–308. DOI: 10.1137/0217017. URL: https://doi.org/10.1137/0217017.

[GMS18] Sanjam Garg, Peihan Miao, and Akshayaram Srinivasan. "Two-Round Multiparty Secure Computation Minimizing Public Key Operations". In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10993. Lecture Notes in Computer Science. Springer, 2018, pp. 273–301. DOI: 10.1007/978-3-319-96878-0_10. URL: https://doi.org/10.1007/978-3-319-96878-0%5C_10.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. Ed. by Alfred V. Aho. ACM, 1987, pp. 218–229. DOI: 10.1145/28395.28420. URL: https://doi.org/10.1145/28395.28420.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004. ISBN: 0-521-83084-2. DOI: 10.1017/CBO9780511721656. URL: http://www.wisdom.weizmann.ac.il/%5C%7Eoded/foc-vol2.html.

[GS17] Sanjam Garg and Akshayaram Srinivasan. "Garbled Protocols and Two-Round MPC from Bilinear Maps". In: *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*. Ed. by Chris Umans. IEEE Computer Society, 2017, pp. 588–599. DOI: 10.1109/FOCS.2017.60. URL: https://doi.org/10.1109/FOCS.2017.60.

[GS18] Sanjam Garg and Akshayaram Srinivasan. "Two-Round Multiparty Secure Computation from Minimal Assumptions". In: *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 468–499. DOI: 10.1007/978-3-319-78375-8_16. URL: https://doi.org/10.1007/978-3-319-78375-8%5C_16.

[Har+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. "On Robust Combiners for Oblivious Transfer and Other Primitives". In: *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*. Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Springer, 2005, pp. 96–113. DOI: 10.1007/11426639_6. URL: https://doi.org/10.1007/11426639%5C_6.

[HK12] Shai Halevi and Yael Tauman Kalai. "Smooth Projective Hashing and Two-Message Oblivious Transfer". In: *J. Cryptology* 25.1 (2012), pp. 158–193. DOI: 10.1007/s00145-010-9092-8. URL: https://doi.org/10.1007/s00145-010-9092-8.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. "Secure Computation on the Web: Computing without Simultaneous Interaction". In: *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*. Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Springer, 2011, pp. 132–150. DOI: 10.1007/978-3-642-22792-9_8. URL: https://doi.org/10.1007/978-3-642-22792-9%5C_8.

[HOZ13] Iftach Haitner, Eran Omri, and Hila Zarosim. "Limits on the Usefulness of Random Oracles". In: *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*. Ed. by Amit Sahai. Vol. 7785. Lecture Notes in Computer Science. Springer, 2013,

pp. 437–456. DOI: 10.1007/978-3-642-36594-2_25. URL: https://doi.org/10.1007/978-3-642-36594-2%5C_25.

[IOS12] Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. "Identifying Cheaters without an Honest Majority". In: *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*. 2012, pp. 21–38.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. "Secure Multi-Party Computation with Identifiable Abort". In: *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*. 2014, pp. 369–386.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. "Founding Cryptography on Oblivious Transfer - Efficiently". In: *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*. Ed. by David A. Wagner. Vol. 5157. Lecture Notes in Computer Science. Springer, 2008, pp. 572–591. DOI: 10.1007/978-3-540-85174-5_32. URL: https://doi.org/10.1007/978-3-540-85174-5%5C_32.

[IR89] Russell Impagliazzo and Steven Rudich. "Limits on the Provable Consequences of One-Way Permutations". In: *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*. Ed. by David S. Johnson. ACM, 1989, pp. 44–61. DOI: 10.1145/73007.73012. URL: https://doi.org/10.1145/73007.73012.

[Jou00] Antoine Joux. "A One Round Protocol for Tripartite Diffie-Hellman". In: *Algorithmic Number Theory, 4th International Symposium, ANTS-IV, Leiden, The Netherlands, July 2-7, 2000, Proceedings*. Ed. by Wieb Bosma. Vol. 1838. Lecture Notes in Computer Science. Springer, 2000, pp. 385–394. DOI: 10.1007/10722028_23. URL: https://doi.org/10.1007/10722028%5C_23.

[Kil88] Joe Kilian. "Founding Cryptography on Oblivious Transfer". In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. Ed. by Janos Simon. ACM, 1988, pp. 20–31. DOI: 10.1145/62212.62215. URL: https://doi.org/10.1145/62212.62215.

[Kus89] Eyal Kushilevitz. "Privacy and Communication Complexity". In: *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*. IEEE Computer Society, 1989, pp. 416–421. DOI: 10.1109/SFCS.1989.63512. URL: https://doi.org/10.1109/SFCS.1989.63512.

[LOZ18] Yehuda Lindell, Eran Omri, and Hila Zarosim. "Completeness for Symmetric Two-Party Functionalities: Revisited". In: *J. Cryptology* 31.3 (2018), pp. 671–697.

[LP09] Yehuda Lindell and Benny Pinkas. "A Proof of Security of Yao's Protocol for Two-Party Computation". In: *J. Cryptology* 22.2 (2009), pp. 161–188. DOI: 10.1007/s00145-008-9036-8. URL: https://doi.org/10.1007/s00145-008-9036-8.

[MMP14] Mohammad Mahmoody, Hemanta K. Maji, and Manoj Prabhakaran. "Limits of random oracles in secure computation". In: *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*. Ed. by Moni Naor. ACM, 2014, pp. 23–34. DOI: 10.1145/2554797.2554801. URL: https://doi.org/10.1145/2554797.2554801.

[MW16] Pratyay Mukherjee and Daniel Wichs. "Two Round Multiparty Computation via Multi-key FHE". In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 735–763. DOI: 10.1007/978-3-662-49896-5_26. URL: https://doi.org/10.1007/978-3-662-49896-5%5C_26.

[NP01] Moni Naor and Benny Pinkas. "Efficient oblivious transfer protocols". In: *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*. Ed. by S. Rao Kosaraju. ACM/SIAM, 2001, pp. 448–457. URL: http://dl.acm.org/citation.cfm?id=365411.365502.

[PS16] Chris Peikert and Sina Shiehian. "Multi-key FHE from LWE, Revisited". In: *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*. Ed. by Martin Hirt and Adam D. Smith. Vol. 9986. Lecture Notes in Computer Science. 2016, pp. 217–238. DOI: 10.1007/978-3-662-53644-5_9. URL: https://doi.org/10.1007/978-3-662-53644-5%5C_9.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. "A Framework for Efficient and Composable Oblivious Transfer". In: *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*. Ed. by David A. Wagner. Vol. 5157. Lecture Notes in Computer Science. Springer, 2008, pp. 554–571. DOI: 10.1007/978-3-540-85174-5_31. URL: https://doi.org/10.1007/978-3-540-85174-5%5C_31.

[Rab81] M. Rabin. *How to exchange secrets by oblivious transfer*. Tech. rep. TR-81. Harvard Aiken Computation Laboratory, 1981.

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. "Notions of Reducibility between Cryptographic Primitives". In: *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*. Ed. by Moni Naor. Vol. 2951. Lecture Notes in Computer

Science. Springer, 2004, pp. 1–20. DOI: 10.1007/978-3-540-24638-1_1. URL: https://doi.org/10.1007/978-3-540-24638-1%5C_1.

[Yao82] Andrew Chi-Chih Yao. "Protocols for Secure Computations (Extended Abstract)". In: *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*. IEEE Computer Society, 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38. URL: https://doi.org/10.1109/SFCS.1982.38.

[Yao86] Andrew Chi-Chih Yao. "How to Generate and Exchange Secrets (Extended Abstract)". In: *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*. IEEE Computer Society, 1986, pp. 162–167. DOI: 10.1109/SFCS.1986.25. URL: https://doi.org/10.1109/SFCS.1986.25.