Two-Round Multiparty Secure Computation from Minimal Assumptions[*]

Sanjam Garg
University of California, Berkeley
sanjamg@berkeley.edu

Akshayaram Srinivasan
University of California, Berkeley
akshayaram@berkeley.edu

Abstract

We provide new two-round multiparty secure computation (MPC) protocols assuming the minimal assumption that two-round oblivious transfer (OT) exists. If the assumed two-round OT protocol is secure against semi-honest adversaries (in the plain model) then so is our two-round MPC protocol. Similarly, if the assumed two-round OT protocol is secure against malicious adversaries (in the common random/reference string model) then so is our two-round MPC protocol. Previously, two-round MPC protocols were only known under relatively stronger computational assumptions. Finally, we provide several extensions.

1 Introduction

Can a group of n mutually distrusting parties compute a joint function of their private inputs without revealing anything more than the output to each other? This is the classical problem of secure computation in cryptography. Yao [Yao86] and Goldreich, Micali and Wigderson [GMW87] provided protocols for solving this problem in the two-party (2PC) and the multiparty (MPC) cases, respectively.

A remarkable aspect of the 2PC protocol based on Yao's garbled circuit construction is its simplicity and the fact that it requires only two-rounds of communication. Moreover, this protocol can be based just on the minimal assumption that two-round 1-out-of-2 oblivious transfer (OT) exists. Two-round OT can itself be based on a variety of computational assumptions such as the Decisional Diffie-Hellman Assumption [AIR01, NP01, PVW08], quadratic residuosity assumption [HK12, PVW08] or the learning-with-errors assumption [PVW08].

In contrast, much less is known about the assumptions that two-round MPC can be based on (constant-round MPC protocols based on any OT protocol are well-known [BMR90]). In particular, two-round MPC protocols are only known under assumptions such as indistinguishability obfuscation [GGHR14, GGH$^+$13] (or, witness encryption [GLS15, GGSW13]), LWE [CM15, MW16, BP16, PS16], or bilinear maps [GS17, BF01, Jou04]. In summary, there is a significant gap between assumptions known to be sufficient for two-round MPC and the assumptions that known to be sufficient for two-round 2PC (or, two-round OT). This brings us to the following main question:

What are the minimal assumptions under which two-round MPC can be constructed?

[*]Research supported in part from AFOSR YIP Award, DARPA/ARL SAFEWARE Award W911NF15C0210, AFOSR Award FA9550-15-1-0274, and research grants by the Okawa Foundation, Visa Inc., and Center for Long-Term Cybersecurity (CLTC, UC Berkeley). The views expressed are those of the author and do not reflect the official policy or position of the funding agencies.

1.1 Our Result

In this work, we give two-round MPC protocols assuming only the necessary assumption that two-round OT exists. In a bit more detail, our main theorem is:

Theorem 1.1 (Main Theorem) *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.*

Previously, such compilers [GGHR14, GLS15, GS17] were only known under comparatively stronger computational assumptions such as indistinguishability obfuscation [BGI$^+$01, GGH$^+$13], witness encryption [GGSW13], or using bilinear maps [GS17, BF01, Jou04]. Additionally, two-round MPC protocols assuming the learning-with-errors assumptions were known [MW16, PS16, BP16] in the CRS model satisfying semi-malicious security.[1] 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 1.2 *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.*

Previously, two-round plain model semi-honest MPC protocols were only known assuming indistinguishability obfuscation [BGI$^+$01, GGH$^+$13], or witness encryption [GGSW13] or bilinear maps [GS17] or from DDH for a constant number of parties [BGI17]. Thus, using two-round plain model OT [NP01, AIR01, HK12] based on standard number theoretic assumptions such as DDH or QR, 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 1.3 *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 DDH based two-round OT protocol (e.g., [PVW08]), this work yields the first two-round malicious MPC protocol in the common random string model under the DDH assumption.

Extensions. In addition to the above main results we obtain several extensions and refer the reader to the main body for details.

[1]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. [AJL$^+$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).

2 Technical Overview

Towards demonstrating the intuition behind our result, in this section, we show how to squish the round complexity of a very simple "toy" protocol to two. Additionally, we sketch how these ideas extend to the general setting and also work in the malicious case. We postpone the details to later sections.

Background: "Garbled Circuits that talk." The starting point of this work is a recent work of Garg and Srinivasan [GS17] that obtains constructions of two-round MPC from bilinear maps. Building on [GGHR14, GLS15], the key idea behind [GS17] is a new method for enabling "garbled circuits to talk," which the authors call "garbled protocols." It is natural to imagine how "garbled circuits that can talk" might be useful for squishing the round complexity of any protocol. By employing this technique, a party can avoid multiple rounds of interaction just by sending a garbled circuit that interacts with the other parties on its behalf. At a technical level, a garbled circuit can "speak" by just outputting a value. However, the idea of enabling garbled circuits to "listen" without incurring any additional interaction poses new challenges. A bit more precisely, "listen" means that a garbled circuit can take as input a bit obtained via a joint computation on its secret state and the secret states of two or more other parties.

In [GS17], this idea was implemented by constructing a special purpose [GOVW12, CDG$^+$17, DG17] witness encryption [GGSW13, BH15] using specific algebraic properties of non-interactive zero-knowledge (NIZK) proofs by Gorth, Ostrovsky and Sahai [GOS06]. The key contribution of this work is a realization of the intuition of "garbled circuits that talk" using any two-round OT protocols rather than a specific NIZK proof system. In particular, we avoid using any specialized algebraic properties of the underlying primitives. At the heart of our construction is the following novel use of two-round OT protocols: in our MPC protocol multiple instances of the underlying two-round OT protocol are executed and the secret receiver's random coins used in some of these executed OT instances are revealed to the other parties. As we explain later, this is done carefully so that the security of the MPC protocol is not jeopardized.

A "toy" protocol for successive ANDs. Stripping away technical details, we highlight our core new idea in the context of a "toy" example, where a garbled circuit will need to listen to one bit. Later, we briefly sketch how this core idea can be used to squish the round complexity of any arbitrary round MPC protocol to two. Recall that, in one round, each party sends a message depending on its secret state and the messages received in prior rounds.

Consider three parties P_1, P_2 and P_3 with inputs α, β, and γ (which are single bits), respectively. Can we realize a protocol such that the parties learn $f(\alpha, \beta, \gamma) = (\alpha, \alpha \wedge \beta, \alpha \wedge \beta \wedge \gamma)$ and nothing more? Can we realize a two-round protocol for the same task? Here is a very simple three-round information theoretic protocol Φ (in the semi-honest setting) for this task: *In the first round*, P_1 sends its input α to P_2 and P_3. *In the second round*, P_2 computes $\delta = \alpha \wedge \beta$ and sends it to P_1 and P_3. Finally, *in the third round*, P_3 computes $\gamma \wedge \delta$ and sends it to P_1 and P_2.

Compiling Φ into a two-round protocol. The key challenge that we face is that the third party's message depends on the second party's message, and the second party's message depends on the first party's message. We will now describe our approach to overcome this three-way dependence using two-round oblivious transfer and thus squish this protocol Φ into a two-round protocol.

We assume the following notation for a two-round OT protocol. *In the first round*, the receiver with choice bit β generates $c = \mathsf{OT}_1(\beta; \omega)$ using ω as the randomness and passes c to the sender. Then *in the second round*, the sender responds with its OT response $d = \mathsf{OT}_2(c, s_0, s_1)$ where s_0 and s_1 are its input strings. *Finally,* using the OT response d and its randomness ω, the receiver recovers s_β. In our protocol below, we will use a circuit $C[\gamma]$ that has a bit γ hardwired in it and that on input a bit δ outputs $\gamma \wedge \delta$. At a high level in our protocol, we will have P_2 and P_3 send extra messages in the first and the second rounds, respectively, so that the third round can be avoided. Here is our protocol:

- **Round 1:** P_1 sends α to P_2 and P_3. P_2 prepares $c_0 = \mathsf{OT}_1(0 \wedge \beta; \omega_0)$ and $c_1 = \mathsf{OT}_1(1 \wedge \beta; \omega_1)$ and sends (c_0, c_1) to P_2 and P_3.
- **Round 2:** P_2 sends $(\alpha \wedge \beta, \omega_\alpha)$ to P_1 and P_3. P_3 garbles $\mathsf{C}[\gamma]$ obtaining $\tilde{\mathsf{C}}$ and input labels lab_0 and lab_1. It computes $d = \mathsf{OT}_2(c_\alpha, \mathsf{lab}_0, \mathsf{lab}_1)$ and sends $(\tilde{\mathsf{C}}, d)$ to P_1 and P_2.
- **Output Evaluation:** Every party recovers lab_δ where $\delta = \alpha \wedge \beta$ from d using ω_α. Next, it evaluates the garbled circuit $\tilde{\mathsf{C}}$ using lab_δ which outputs $\gamma \wedge \delta$ as desired.

Intuitively, in the protocol above P_2 sends two first OT messages c_0 and c_1 that are prepared assuming α is 0 and assuming α is 1, respectively. Note that P_3 does not know α at the beginning of the first round, but P_3 does know it at the end of the first round. Thus, P_3 just uses c_α while discarding $c_{1-\alpha}$ in preparing its messages for the second round. This achieves the three-way dependency while only using two-rounds. Furthermore, P_2's second round message reveals the randomness ω_α enabling all parties (and not just P_2 and P_3) to obtain the label lab_δ which can then be used for evaluation of $\tilde{\mathsf{C}}$. In summary, via this mechanism, the garbled circuit $\tilde{\mathsf{C}}$ was able to "listen" to the bit δ that P_3 did not know when generating the garbled circuit.

The above description highlights our ideas for squishing round complexity of an incredibly simple toy protocol where only one bit was being "listened to." Moreover, the garbled circuit "speaks" or outputs $\gamma \wedge \delta$, which is obtained by all parties. In the above "toy" example, P_3's garbled circuit computes a gate that takes only one bit as input. To compute a gate with two bit inputs, P_2 will need to send four first OT messages in the first round instead of two.

Squishing arbitrary protocols. Our approach to enable garbled circuits to "listen to" a larger number of bits with complex dependencies is as follows. We show that any MPC protocol Φ between parties $P_1, \cdots P_n$ can be transformed into one satisfying the following format. First, the parties execute a pre-processing step; namely, each party P_i computes some randomized function of its input x_i obtaining public value z_i which is shared with everyone else and private value v_i. z_i is roughly an encryption of x_i using randomness from v_i as a one-time pad. v_i also contains random bits that will be used as one-time pad to encrypt bits sent later by P_i. *Second*, each party sets its local state $\mathsf{st}_i = (z_1 \| \ldots \| z_n) \oplus v_i$. That places us at the beginning of the protocol execution phase. In our transformed protocol Φ can be written as a sequence of T *actions*. For each $t \in [T]$ the t^{th} action $\phi_t = (i, f, g, h)$ involves party P_i computing *one* NAND gate; it sets $\mathsf{st}_{i,h} = \mathsf{NAND}(\mathsf{st}_{i,f}, \mathsf{st}_{i,g})$ and sends $v_{i,h} \oplus \mathsf{st}_{i,h}$ to all the other parties. Our transformed protocol is such that for any bit $\mathsf{st}_{i,h}$, the bit $v_{i,h}$ is unique and acts as the one-time pad to hide it from the other parties. (Some of the bits in v_i are set to 0. These bits do not need to be hidden from other parties.) *To complete this action*, each party P_j for $j \neq i$ sets $\mathsf{st}_{j,h}$ to be the received bit. After all the actions are completed, each party P_j outputs a function of its local state st_j. In this transformed MPC protocol, in any round only one bit is sent based on just one gate (i.e., the gate obtained as $v_{i,h} \oplus \mathsf{NAND}(\mathsf{st}_{i,f}, \mathsf{st}_{i,g})$

with inputs $\mathsf{st}_{i,f}$ and $\mathsf{st}_{i,g}$, where $v_{i,h}$ is hardwired inside it) computation on two bits. Thus, we can use the above "toy" protocol to achieve this effect.

To squish the round complexity of this transformed protocol, *in the first round*, we will have each party follow the pre-processing step from above along with a bunch of carefully crafted first OT messages as in our "toy" protocol. *In the second round*, parties will send a garbled circuit that is expected to "speak" and "listen" to the garbled circuits of the other parties. So when $\phi_1 = (i, f, g, h)$ is executed, we have that the garbled circuit sent by party P_i speaks and all the others listen. Each of these listening garbled circuits uses our "toy" protocol idea from above. After completion of the first action, all the garbled circuits will have read the transcript of communication (which is just the one bit communicated in the first action ϕ_1). Next, the parties need to execute action $\phi_2 = (i, f, g, h)$ and this is done like the first action, and the process continues. This completes the main idea of our construction. Building on this idea, we obtain a compiler that assuming semi-honest two-round OT transforms any semi-honest MPC protocol into a two-round semi-honest MPC protocol. Furthermore, if the assumed semi-honest two-round OT protocol is in the plain model then so will be the resulting MPC protocol.

Compilation in the Malicious Case. The protocol ideas described above only achieve semi-honest security and additional use of non-interactive zero-knowledge (NIZK) proofs [BFM88, FLS90] is required to upgrade security to malicious [AJL$^+$12, MW16]. This has been the case for all known two-round MPC protocol constructions. In a bit more detail, by using NIZKs parties can (without increasing the round complexity) prove in zero-knowledge that they are following protocol specifications. The use of NIZKs might seem essential to such protocols. However, we show that this can be avoided. Our main idea is as follows: instead of proving that the garbled circuits are honestly generated, we require that the garbled circuits prove to each other that the messages they send are honestly generated. Since our garbled circuits can "speak" and "listen" over several rounds without increasing the round complexity of the squished protocol, therefore we can instead use interactive zero-knowledge proof system and avoid NIZKs. Building on this idea we obtain two-round MPC protocols secure against malicious adversaries. We elaborate on this new idea and other issues involved in subsequent sections.

3 Preliminaries

We recall some standard cryptographic definitions in this section. Let λ denote the security parameter. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for any polynomial $\mathsf{poly}(\cdot)$ there exists λ_0 such that for all $\lambda > \lambda_0$ we have $\mu(\lambda) < \frac{1}{\mathsf{poly}(\lambda)}$. We will use $\mathsf{negl}(\cdot)$ to denote an unspecified negligible function and $\mathsf{poly}(\cdot)$ to denote an unspecified polynomial function.

For a probabilistic algorithm A, we denote $A(x; r)$ to be the output of A on input x with the content of the random tape being r. When r is omitted, $A(x)$ denotes a distribution. For a finite set S, we denote $x \leftarrow S$ as the process of sampling x uniformly from the set S. We will use PPT to denote Probabilistic Polynomial Time algorithm.

3.1 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 $(\mathsf{Garble}, \mathsf{Eval})$. Garble is the circuit garbling procedure and Eval is the corresponding evaluation procedure. More formally:

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

- $y \leftarrow \mathsf{Eval}\left(\widetilde{\mathsf{C}}, \{\mathsf{lbl}_{w,x_w}\}_{w \in \mathsf{inp}(C)}\right)$: Given a garbled circuit $\widetilde{\mathsf{C}}$ and a sequence of input labels $\{\mathsf{lbl}_{w,x_w}\}_{w \in \mathsf{inp}(C)}$ (referred to as the garbled input), Eval outputs a string y.

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

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

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

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

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

where $(\widetilde{\mathsf{C}}, \{\mathsf{lbl}_{w,b}\}_{w \in \mathsf{inp}(C), b \in \{0,1\}}) \leftarrow \mathsf{Garble}\left(1^\lambda, C\right)$ and $\overset{c}{\approx}$ denotes that the two distributions are computationally indistinguishable.

3.2 Universal Composability Framework

We work in the the Universal Composition (UC) framework [Can01] to formalize and analyze the security of our protocols. (Our protocols can also be analyzed in the stand-alone setting, using the composability framework of [Can00a, Gol04], or in other UC-like frameworks, like that of [PW00].) We provide a brief overview of the framework in Appendix A and refer the reader to [Can00b] for details.

3.3 Oblivious Transfer

In this paper, we consider a 1-out-of-2 *oblivious transfer* protocol (OT), similar to [CCM98, NP01, AIR01, DHRS04, 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.

Security of the oblivious transfer (OT) functionality can be described easily by an ideal functionality $\mathcal{F}_{\mathsf{OT}}$ as is done in [CLOS02]. However, in our constructions the receiver needs to reveal the randomness (or a part of the randomness) it uses in an instance of two-round OT to other parties. Therefore, defining security as an ideal functionality raises issues require care and issues similar to one involved in defining ideal public-key encryption functionality [Can05, Page 96] arise. Thus, in our context, it is much easier to directly work with a two-round OT protocol. We define the syntax and the security guarantees of a two-round OT protocol below.

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 $(\mathsf{ots}_1, \omega) \leftarrow \mathsf{OT}_1(1^\lambda, \beta)$, $\mathsf{ots}_2 \leftarrow \mathsf{OT}_2(\mathsf{ots}_1, (s_0, s_1))$, then $\mathsf{OT}_3(\mathsf{ots}_2, (\beta, \omega)) = s_\beta$ with overwhelming probability.

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

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

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

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

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

Constructions of semi-honest two-round OT are known in the plain model under assumptions such as DDH [AIR01, NP01] and quadratic 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.

- **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 $\sigma \leftarrow K_{\mathsf{OT}}(1^\lambda)$, $(\mathsf{ots}_1, \omega) \leftarrow \mathsf{OT}_1(\sigma, \beta)$, $\mathsf{ots}_2 \leftarrow \mathsf{OT}_2(\sigma, \mathsf{ots}_1, (s_0, s_1))$, then $\mathsf{OT}_3(\sigma, \mathsf{ots}_2, (\beta, \omega)) = s_\beta$ with overwhelming probability.

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

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

- **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

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

[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.

```
**Experiment** $\mathrm{IND}^{\mathrm{REAL}}_{\mathcal{A}}(1^\lambda, K_0, K_1)$:
    $\sigma \leftarrow K_{\mathsf{OT}}(1^\lambda)$
    $\mathsf{ots}_1 \leftarrow \mathcal{A}(\sigma)$
    $\mathsf{ots}_2 \leftarrow \mathsf{OT}_1(\sigma, \mathsf{ots}_1, (K_0, K_1))$
    Output $\mathcal{A}(\mathsf{ots}_2)$
```

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

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 $\beta \in \{0,1\}$:

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

Using standard techniques in the literature (e.g., [CLOS02]) it is possible to add equivocal receiver's security to any OT protocol. We sketch a construction in Appendix B for the sake of completeness.

4 Conforming Protocols

Our protocol compilers work for protocols satisfying certain syntactic structure. 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.

4.1 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 $k \in [\ell] \setminus \{(i-1)\ell/n + 1, \ldots, i\ell/n\}$.

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

- **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:

  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 $\mathsf{post}(\mathsf{st}_i)$.

4.2 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 4.1 *Any MPC protocol* Π *can be written as 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. 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. ■

5 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.

5.1 Our Compiler

We give our construction of two-round MPC in Figure 1 and the circuit that needs to be garbled (repeatedly) is shown in Figure 2. We start by providing intuition behind this construction.

5.1.1 Overview

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 $\gamma := \mathsf{NAND}(\mathsf{st}^t_{i^*,f}, \mathsf{st}^t_{i^*,g})$, writes this bit to position h of $\mathsf{st}^t_{i^*}$ 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^t_i.

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^t_i 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} garbled circuits in each of the n sequences 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 $\mathsf{st}^t_{i^*}$, updates the local state by writing the bit γ in the position h of $\mathsf{st}^t_{i^*}$ 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. This was achieved in [GS17] by using specific properties of Groth, Ostrovsky and Sahai proofs and in this work, we only rely on oblivious transfer. This is our key new idea and we provide the details next.

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 $(\mathsf{label}^{i,t+1}_{h,0}, \mathsf{label}^{i,t+1}_{h,1})$) 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 $\mathsf{label}^{i,t+1}_{h,0}, \mathsf{label}^{i,t+1}_{h,1}$ 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 $\{\mathsf{st}^t_{i,k}\}_{k\in[\ell]\setminus\{h\}}$ (as these bits do not change at the end of the t-th round) and additionally outputs the sender OT response for $\alpha = \mathsf{st}^t_{i,f}$ and $\beta = \mathsf{st}^t_{i,g}$ with the messages being set to the labels corresponding to h-th bit of st^t_i. 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, $\alpha = \mathsf{st}^t_{i^*,f} \oplus v_{i^*,f}, \beta = \mathsf{st}^t_{i^*,g} \oplus v_{i^*,g}$. 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 $\mathsf{label}^{i,t+1}_{h,\gamma\oplus v_{i^*,h}}$ 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 $\mathsf{label}^{i,t+1}_{h,\gamma\oplus v_{i^*,h}}$.

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.

Theorem 5.1 *Let Φ be a polynomial round, n-party semi-honest MPC protocol computing a function $f : (\{0,1\}^m)^n \to \{0,1\}^*$, $(\mathsf{Garble}, \mathsf{Eval})$ 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 1 is a two-round, n-party semi-honest MPC protocol computing f against static corruptions.*

This theorem is proved in the rest of this section.

5.2 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 $\mathsf{NAND}(\mathsf{st}_{i^*,f}, \mathsf{st}_{i^*,g}) \oplus v_{i^*,h}$. 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 $\mathsf{st}_{i,g} = \mathsf{st}_{i^*,g} \oplus v_{i^*,g}$. 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.

Let Φ be an n-party conforming semi-honest MPC protocol, $(\mathsf{Garble}, \mathsf{Eval})$ 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 $\alpha, \beta \in \{0,1\}$
$$\mathsf{ots}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}_1(1^\lambda, v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta); \omega_{t,\alpha,\beta}).$$
3. Send $\left(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t \in A_i, \alpha,\beta \in \{0,1\}}\right)$ 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 $\overline{\mathsf{lab}}^{i,T+1} := \{\mathsf{lab}^{i,T+1}_{k,0}, \mathsf{lab}^{i,T+1}_{k,1}\}_{k \in [\ell]}$ 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 2)
   $$\left(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{lab}}^{i,t}\right) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{P}[i, \phi_t, v_i, \{\omega_{t,\alpha,\beta}\}_{\alpha,\beta}, \bot, \overline{\mathsf{lab}}^{i,t+1}]).$$
   (c) If $i \neq i^*$ then for every $\alpha, \beta \in \{0,1\}$, set $\mathsf{ots}^i_{2,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{ots}_{1,t,\alpha,\beta}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$ and compute
   $$\left(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{lab}}^{i,t}\right) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{P}[i, \phi_t, v_i, \bot, \{\mathsf{ots}^i_{2,t,\alpha,\beta}\}_{\alpha,\beta}, \overline{\mathsf{lab}}^{i,t+1}]).$$
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 $(\mathsf{ots}_2, \{\mathsf{lab}^{j,t+1}_k\}_{k \in [\ell] \setminus \{h\}}) := \mathsf{Eval}(\widetilde{\mathsf{P}}^{j,t}, \widetilde{\mathsf{lab}}^{j,t})$.
      ii. Recover $\mathsf{lab}^{j,t+1}_h := \mathsf{OT}_3(\mathsf{ots}_2, \omega)$.
      iii. Set $\widetilde{\mathsf{lab}}^{j,t+1} := \{\mathsf{lab}^{j,t+1}_k\}_{k \in [\ell]}$.
3. Compute the output as $\mathsf{post}(i, \mathsf{st}_i)$.

Figure 1: Two-round Semi-Honest MPC.

```
                                   P

Input. $\mathsf{st}_i$.
Hardcoded. The index $i$ of the party, the action $\phi_t = (i^*, f, g, h)$, the secret value $v_i$, the strings $\{\omega_{t,\alpha,\beta}\}_{\alpha,\beta}$, $\{\mathsf{ots}_{2,t,\alpha,\beta}\}_{\alpha,\beta}$ 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 $(\mathsf{ots}_{2,t,\mathsf{st}_{i,f},\mathsf{st}_{i,g}}, \{\mathsf{lab}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]\setminus\{h\}})$.
```

Figure 2: The program P.

Via the same argument as above 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 $\mathsf{Z} \in \{0,1\}^t$.

5.3 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 simulator.

Description of the Simulator. We give the description of the ideal world adversary $\mathcal{S}$ that simulates the view of the real world adversary $\mathcal{A}$. $\mathcal{S}$ 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 $\mathcal{S}$ receives from $\mathcal{Z}$, $\mathcal{S}$ writes that value to $\mathcal{A}$'s input tape. Similarly, the output of $\mathcal{A}$ is written as the output on $\mathcal{S}$'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**: $\mathcal{S}$ 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 $i \in [n] \setminus H$ $\mathcal{S}$ 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

[4]For simplicity of exposition, we only consider the case where every party gets the same output. The proof in the more general case where parties get different outputs follows analogously.

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 for each $i \in [n]$ and $k \in [\ell]$ is equal to $\mathsf{st}_{i,k} \oplus v_{i,k}$). $\mathcal{S}$ 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 $\mathcal{S}$ to $\mathcal{A}$:** Next $\mathcal{S}$ 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 $\mathsf{ots}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}_1(1^\lambda, \mathsf{Z}_t; \omega_{t,\alpha,\beta})$. For each $i \in H$, $\mathcal{S}$ sends $(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t \in A_i, \alpha,\beta \in \{0,1\}})$ to the adversary $\mathcal{A}$ on behalf of the honest party P_i.

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

- **Round-2 messages from $\mathcal{S}$ to $\mathcal{A}$:** For each $i \in H$, the simulator $\mathcal{S}$ 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
     $$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}_k^{i,t}\}_{k \in [\ell]}\right) \leftarrow \mathsf{Sim}_\mathsf{G}\left(1^\lambda, \left((\alpha^*, \beta^*, \gamma^*), \omega_{t,\alpha^*,\beta^*}, \{\mathsf{lab}_k^{i,t+1}\}_{k \in [\ell]}\right)\right).$$
     (d) If $i \neq i^*$ then set $\mathsf{ots}^i_{2,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2(\mathsf{ots}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}_h^{i,t+1}, \mathsf{lab}_h^{i,t+1})$ and compute
     $$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}_k^{i,t}\}_{k \in [\ell]}\right) \leftarrow \mathsf{Sim}_\mathsf{G}\left(1^\lambda, \left(\mathsf{ots}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{lab}_k^{i,t+1}\}_{k \in [\ell] \setminus \{h\}}\right)\right).$$
  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 $\mathcal{S}$:** For every $i \in [n] \setminus H$, $\mathcal{S}$ obtains the second round message from $\mathcal{A}$ on behalf of the malicious parties. Subsequent to obtaining these messages, for each $i \in H$, $\mathcal{S}$ sends $(\mathsf{generateOutput}, \mathsf{sid}, \{P_1 \cdots P_n\}, P_i)$ to the ideal functionality.

5.4 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 $\mathcal{S}$. We prove this via an hybrid argument with $T + 1$ hybrids.

- $\mathcal{H}_{Real}$: This hybrid is the same as the real world execution. Note that this hybrid is the same as hybrid $\mathcal{H}_t$ below with $t = 0$.

- $\mathcal{H}_t$ (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.

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 st^* be the local state of the end of execution of $\mathsf{Faithful}$. 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. $\mathcal{S}$ makes two changes in how it generates messages on behalf of P_{i^*}. First, for all $\alpha, \beta \in \{0,1\}$, $\mathcal{S}$ generates $\mathsf{ots}_{1,t,\alpha,\beta}$ 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

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

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

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

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

where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]}$ 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 5.2.

- $\mathcal{H}_{T+1}$: 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 these values by executing 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 $\mathcal{H}_T$ and $\mathcal{H}_{T+1}$ follows directly from the semi-honest 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 5.2 *Assuming semi-honest security of the two-round OT protocol and the security of the garbling scheme, for all $t \in \{1 \ldots T\}$ 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^*} be the state of P_{i^*} at the end of round t, and $\alpha^* := \mathsf{st}_{i^*,f} \oplus v_{i^*,f}$, $\beta^* := \mathsf{st}_{i^*,g} \oplus v_{i^*,g}$ and $\gamma^* := \mathsf{st}_{i^*,h} \oplus v_{i^*,h}$. 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 $\mathcal{S}$ 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
  $$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^\lambda, \left((\alpha^*, \beta^*, \gamma^*), \omega_{t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]}\right)\right),$$
  where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]}$ are the honestly generates input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i,t+1}$.
  - If $i \neq i^*$ then
  $$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^\lambda, \left(\mathsf{ots}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]\setminus\{h\}}\right)\right),$$
  where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]}$ are the honestly generated input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i,t+1}$.

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

- $\mathcal{H}_{t,2}$: Skip this hybrid, if $i^* \notin H$. This hybrid is same as $\mathcal{H}_{t,1}$ except that we change how $\mathcal{S}$ generates the Round-1 message on behalf of P_{i^*}. Specifically, the simulator $\mathcal{S}$ generates $\mathsf{ots}_{1,t,\alpha,\beta}$ as is done in the $\mathcal{H}_t$. In a bit more detail,

  for all $\alpha, \beta \in \{0,1\}$, $\mathcal{S}$ generates $\mathsf{ots}_{1,t,\alpha,\beta}$ 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 $\mathcal{H}_{t,1}$ and $\mathcal{H}_{t,2}$ 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 $\beta \neq \beta^*$). This is sketched in Appendix C.

- $\mathcal{H}_{t,3}$: Skip this hybrid if there does not exist $i \neq i^*$ such that $i \in H$. In this hybrid, we change how $\mathcal{S}$ generates the $\mathsf{ots}^i_{2,t,\alpha,\beta}$ 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, $\mathcal{S}$ only generates one of these four values; namely, $\mathsf{ots}^i_{2,t,\alpha^*,\beta^*}$ which is now generated as $\mathsf{OT}_2(\mathsf{ots}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t})$ instead of $\mathsf{OT}_2(\mathsf{ots}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$.

  Indistinguishability between hybrids $\mathcal{H}_{t,2}$ and $\mathcal{H}_{t,3}$ follows directly from the sender's security of underlying semi-honest oblivious transfer protocol. Finally, observe that $\mathcal{H}_{t,3}$ is the same as hybrid $\mathcal{H}_t$.

■

5.5 Extensions

The protocol presented above is very general and can be extended in different ways to obtain several other additional properties. We list some of the simple extensions below.

Multi-Round OT. We note that plugging in any multi-round (say, r-round) OT scheme with semi-honest security we obtain an r-round MPC for semi-honest adversaries. More specifically, this can be achieved as follows. We run the first $r-2$ rounds of the protocol as a pre-processing phase with the receiver's choice bits set as in the protocol and the sender's message being randomly chosen labels. We then run the first round of our MPC protocol with the $(r-1)^{th}$ round of OT from the receiver and run the second round using the last round message from the sender hardwired inside the garbled circuits. The proof of security follows identically to proof given above for a two-round OT. A direct corollary of this construction is a construction of three round MPC for semi-honest adversaries from enhanced trapdoor permutations.

Two-Round MPC for RAM programs. In the previous section, we described how protocol compilation can be done for the case of conforming MPC protocols for circuits. Specifically, the protocol communication depends on the lengths of the secret state of the parties. We note that we can extend this framework for securely evaluating RAM programs [OS97, GKK^{+}12, LO13, GHL^{+}14, GLOS15, GLO15, GGMP16, HY16] in two-rounds. In this setting, each party has a huge database as its private input and the parties wishes to compute a RAM program on their private databases. We consider the persistent memory setting [LO13, GHL^{+}14] where several programs are evaluated on the same databases. We allow an (expensive) pre-processing phase where the parties communicate to get a shared garbled database and the programs must be evaluated with communication and computation costs that grow with the running time of the programs. In our construction of two-round MPC for RAM programs, the pre-processing phase involves the parties executing a two-round MPC to obtain garbled databases of all the parties using a garbled RAM scheme (say, [GLOS15]) along with the shared secret state. Next, when a program needs to be executed, then the parties execute our two-round MPC to obtain a garbled program. Finally, the obtained garbled program can be executed with the garbled database to obtain the output.

Reducing the Communication Complexity. Finally, we note that in our two-round protocol each party can reduce the communication complexity [Gen09, BGI16, CDG^{+}17] of either one of its two messages (with size dependent just on the security parameter) using Laconic Oblivious Transfer (OT) [CDG^{+}17]. Roughly, laconic OT allows one party to commit to a large message by a short *hash* string (depending just on the security parameter) such that the knowledge of the laconic hash suffices for generating a garbled circuit that can be executed on the large committed string as input. Next, we give simple transformations using which the first party in any two-round MPC protocol can make either its first message or its second message short, respectively. The general case can also be handled in a similar manner.

We start by providing a transformation by which the first party can make its first message short. The idea is that in the transformed protocol the first party now only sends a laconic hash of the first message of the underlying protocol, which is disclosed in the second round message of the transformed protocol. The first round of messages of all other parties in the transformed protocol remains unchanged. However, their second round messages are now obtained by sending garbled circuits that generate the second round message of the original protocol using the first round message of the first party as input. This can be done using laconic OT.

Using a similar transformation the first party can make its second message short. Specifically, in this case, the first party appends its first round message with a garbled circuit that generated its second round message given as input the laconic OT hash for the first round messages of all the other parties. Now in the second round, the first party only needs to disclose the labels for the garbled circuit corresponding to laconic OT hash of the first round messages of all the other parties. The messages of all the other parties remain unchanged.

6 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 as argued earlier can be added with need for any additional assumptions) This is achieved by designing a compiler that takes any conforming arbitrary (polynomial) round MPC protocol Φ and squishes it to two rounds.

6.1 Our Compiler

We give our construction of two-round MPC in Figure 3 and the circuit that needs to be garbled (repeatedly) is shown in Figure 2 (same as the semi-honest case). We start by providing intuition behind this construction. Our compiler is essentially the same as the semi-honest case. In addition to the minor syntactic changes, the main difference is that we compile malicious secure conforming protocols instead of semi-honest ones.

Another technical issue arises because the adversary may wait to receiver first round messages that $\mathcal{S}$ 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 $\mathcal{S}$ 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 6.1 *Let Φ be a polynomial round, n-party malicious MPC protocol computing a function $f : (\{0,1\}^m)^n \to \{0,1\}^*$, $(\mathsf{Garble}, \mathsf{Eval})$ 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 is a two-round, n-party malicious MPC protocol computing f against static corruptions.*

We prove the security of our compiler in the rest of the section. The proof of correctness is the same as for the case of semi-honest security (see Section 5.2).

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 $\mathsf{Z} \in \{0,1\}^t$.

6.2 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.

Let Φ be an n-party conforming malicious MPC protocol, $(\mathsf{Garble}, \mathsf{Eval})$ be a garbling scheme for circuits and $(K_{\mathsf{OT}}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ be a malicious (with equivocal receiver security) two-round oblivious transfer protocol.

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 $\alpha, \beta \in \{0,1\}$

$$\mathsf{ots}_{1,t,\alpha,\beta} \leftarrow \mathsf{OT}_1(\sigma_{t,\alpha,\beta}, v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta); \omega_{t,\alpha,\beta}).$$

3. Send $\left(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}}\right)$ 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 $\overline{\mathsf{lab}}^{i,T+1} := \{\mathsf{lab}^{i,T+1}_{k,0}, \mathsf{lab}^{i,T+1}_{k,1}\}_{k\in[\ell]}$ 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 2)

$$\left(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{lab}}^{i,t}\right) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{P}[i, \phi_t, v_i, \{\omega_{t,\alpha,\beta}\}_{\alpha,\beta}, \bot, \overline{\mathsf{lab}}^{i,t+1}]).$$

   (c) If $i \neq i^*$ then for every $\alpha, \beta \in \{0,1\}$, set $\mathsf{ots}^i_{2,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\sigma_{t,\alpha,\beta}, \mathsf{ots}_{1,t,\alpha,\beta}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$ and compute

$$\left(\widetilde{\mathsf{P}}^{i,t}, \overline{\mathsf{lab}}^{i,t}\right) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{P}[i, \phi_t, v_i, \bot, \{\mathsf{ots}^i_{2,t,\alpha,\beta}\}_{\alpha,\beta}, \overline{\mathsf{lab}}^{i,t+1}]).$$

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 $(\mathsf{ots}_2, \{\mathsf{lab}^{j,t+1}_k\}_{k\in[\ell]\setminus\{h\}}) := \mathsf{Eval}(\widetilde{\mathsf{P}}^{j,t}, \widetilde{\mathsf{lab}}^{j,t})$.
      ii. Recover $\mathsf{lab}^{j,t+1}_h := \mathsf{OT}_3(\sigma_{t,\alpha,\beta}, \mathsf{ots}_2, \omega)$.
      iii. Set $\widetilde{\mathsf{lab}}^{j,t+1} := \{\mathsf{lab}^{j,t+1}_k\}_{k\in[\ell]}$.
3. Compute the output as $\mathsf{post}(i, \mathsf{st}_i)$.

Figure 3: Two-round Malicious MPC.

Faithful Execution. In the first round of our compiled protocol, $\mathcal{A}$ provides z_i for every $i \in [n] \backslash H$ and $\mathsf{ots}_{1,t,\alpha,\beta}$ for every $t \in \cup_{i \in [n] \backslash h}$ 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 $\mathsf{ots}_{1,t,\alpha,\beta}$. Intuitively speaking, a faithful execution is an execution that is consistent with these extracted values.

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 $\mathsf{st}^* := b_{t,\mathsf{st}^*_f,\mathsf{st}^*_g}$ 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 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 $\mathcal{S}$ that simulates the view of the real world adversary $\mathcal{A}$. $\mathcal{S}$ 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 $\mathcal{S}$ receives from $\mathcal{Z}$, $\mathcal{S}$ writes that value to $\mathcal{A}$'s input tape. Similarly, the output of $\mathcal{A}$ is written as the output on $\mathcal{S}$'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**: $\mathcal{S}$ generates the common random/reference string as follows:
  1. For each $i \in H, t \in A_i, \alpha, \beta \in \{0,1\}$ set $(\sigma_{t,\alpha,\beta}, (\mathsf{ots}_{1,t,\alpha,\beta}, \omega^0_{t,\alpha,\beta}, \omega^1_{t,\alpha,\beta})) \leftarrow \mathsf{Sim}_{Eq}(1^\lambda)$ (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**: $\mathcal{S}$ executes the simulator (against malicious adversary's) $\mathsf{Sim}_\Phi(1^\lambda)$ to obtain $\{z_i\}_{i\in H}$. Moreover, $\mathcal{S}$ starts the real-world adversary $\mathcal{A}$. We next describe how $\mathcal{S}$ provides its messages to Sim_Φ and $\mathcal{A}$.

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

- **Round-1 messages from $\mathcal{A}$ to $\mathcal{S}$:** Corresponding to every $i \in [n]\setminus H$, $\mathcal{S}$ receives from the adversary $\mathcal{A}$ the value $(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t\in A_i,\alpha,\beta\in\{0,1\}})$ 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 $b_{t,\alpha,\beta} := \mathsf{Ext}_2(\tau_{t,\alpha,\beta}, \mathsf{ots}_{1,t,\alpha,\beta})$.

- **Completing the execution with the Sim_Φ**: For each $i \in [n]\setminus H$, $\mathcal{S}$ 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. $\mathcal{S}$ provides computation phase messages to Sim_Φ by following a faithful execution. More formally, for every corrupted party P_i where $i \in [n]\setminus H$, $\mathcal{S}$ 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 $i \in [n]\setminus H$, $\mathcal{S}$ 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 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 $\mathcal{S}$ to $\mathcal{A}$:** For each $i \in H$, the simulator $\mathcal{S}$ 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
     $$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^\lambda, \left((\alpha^*, \beta^*, \gamma^*), \omega_{t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_k\}_{k\in[\ell]}\right)\right).$$
     (d) If $i \neq i^*$ then set $\mathsf{ots}^i_{2,t,\alpha^*,\beta^*} \leftarrow \mathsf{OT}_2(\sigma_{t,\alpha^*,\beta^*}, \mathsf{ots}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_h, \mathsf{lab}^{i,t+1}_h)$ and compute
     $$\left(\widetilde{\mathsf{P}}^{i,t}, \{\mathsf{lab}^{i,t}_k\}_{k\in[\ell]}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^\lambda, \left(\mathsf{ots}^i_{2,t,\alpha^*,\beta^*}, \{\mathsf{lab}^{i,t+1}_k\}_{k\in[\ell]\setminus\{h\}}\right)\right).$$
  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 $\mathcal{S}$:** For every $i \in [n]\setminus H$, $\mathcal{S}$ obtains the second round message from $\mathcal{A}$ on behalf of the malicious parties. Subsequent to obtaining these messages, $\mathcal{S}$ 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 for each $i \in H$, $\mathcal{S}$ sends $(\mathsf{generateOutput}, \mathsf{sid}, \{P_1 \cdots P_n\}, P_i)$ to the ideal functionality.

6.3 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 $\mathcal{S}$. We prove this via an hybrid argument with $T + 2$ 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 is generated as is done in the simulation. More formally, $\mathcal{S}$ generates the common random/reference string as follows:
  1. For each $i \in H, t \in A_i, \alpha, \beta \in \{0,1\}$ set $(\sigma_{t,\alpha,\beta}, (\mathsf{ots}_{1,t,\alpha,\beta}, \omega^0_{t,\alpha,\beta}, \omega^1_{t,\alpha,\beta})) \leftarrow \mathsf{Sim}_{Eq}(1^\lambda)$ (using equivocal simulator).

     For all $t \in \cup_{i\in H} A_i$ and $\alpha, \beta \in \{0,1\}$ set $\omega_{t,\alpha,\beta} := \omega_{t,\alpha,\beta}^{v_{i,h} \oplus \mathsf{NAND}(v_{i,f}\oplus\alpha, v_{i,g}\oplus\beta)}$ where v_i is the secret value of party P_i generated in the pre-processing phase of Φ.
  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).

     Corresponding to every $i \in [n] \setminus H$, $\mathcal{A}$ sends $(z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t\in A_i, \alpha,\beta\in\{0,1\}})$ 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 $b_{t,\alpha,\beta} := \mathsf{Ext}(\tau_{t,\alpha,\beta}, \mathsf{ots}_{1,t,\alpha,\beta})$.

  Note that this hybrid is the same as hybrid $\mathcal{H}_t$ below with $t = 0$.

  The indistinguishability between hybrids $\mathcal{H}_{Real}$ and $\mathcal{H}_0$ follow from a reduction to the sender's security and the equivocal receiver'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 $\mathcal{S}$ (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, $\mathcal{S}$ 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, $\mathcal{S}$ 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 st^* be the local state of the end of execution of $\mathsf{Faithful}$. 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. $\mathcal{S}$ makes two changes in how it generates messages on behalf of P_{i^*}. First, for all $\alpha, \beta \in \{0,1\}$, $\mathcal{S}$ sets $\omega_{t,\alpha,\beta}$ as $\omega^{\mathsf{Z}_t}_{t,\alpha,\beta}$ rather than $\omega_{t,\alpha,\beta}^{v_{i,h}\oplus\mathsf{NAND}(v_{i,f}\oplus\alpha, v_{i,g}\oplus\beta)}$ (note that these two values are the same when using the honest party's input and randomness). Second, it generates the garbled circuit

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

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

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

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

where $\{\mathsf{lab}^{i,t+1}_{k,\mathsf{st}_{i,k}}\}_{k\in[\ell]}$ 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 5.2.

- $\mathcal{H}_{T+1}$: 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 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 $\mathcal{H}_T$ and $\mathcal{H}_{T+1}$ 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 6.2 *Assuming malicious 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^*} be the state of P_{i^*} at the end of round t, and $\alpha^* := \mathsf{st}_{i^*,f} \oplus v_{i^*,f}$, $\beta^* := \mathsf{st}_{i^*,g} \oplus v_{i^*,g}$ and $\gamma^* := \mathsf{st}_{i^*,h} \oplus v_{i^*,h}$. 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 $\mathcal{S}$ 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

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

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

  - If $i \neq i^*$ then

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

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

The indistinguishability between hybrids $\mathcal{H}_{t,1}$ and $\mathcal{H}_{t-1}$ follows by $|H|$ invocations of security of the garbling scheme. The reduction is analogous to the semi-honest case.

- $\mathcal{H}_{t,2}$: Skip this hybrid, if $i^* \notin H$. This hybrid is same as $\mathcal{H}_{t,1}$ except that we change how $\mathcal{S}$ generates the Round-1 message on behalf of P_{i^*}. Specifically, the simulator $\mathcal{S}$ generates $\mathsf{ots}_{1,t,\alpha,\beta}$ as is done in the $\mathcal{H}_t$. In a bit more detail,

  for all $\alpha, \beta \in \{0,1\}$, $\mathcal{S}$ generates $\mathsf{ots}_{1,t,\alpha,\beta}$ as $\mathsf{OT}_1(\sigma_{t,\alpha,\beta}, \mathsf{Z}_t; \omega_{t,\alpha,\beta})$ rather than $\mathsf{OT}_1(\sigma t, \alpha, \beta, v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta); \omega_{t,\alpha,\beta})$.

  Indistinguishability between hybrids $\mathcal{H}_{t,1}$ and $\mathcal{H}_{t,2}$ 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 $\beta \neq \beta^*$). This reduction is exactly similar to the semi-honest case.

- $\mathcal{H}_{t,3}$: Skip this hybrid if there does not exist $i \neq i^*$ such that $i \in H$. In this hybrid, we change how $\mathcal{S}$ generates the $\mathsf{ots}^i_{2,t,\alpha,\beta}$ 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, $\mathcal{S}$ only generates one of these four values; namely, $\mathsf{ots}^i_{2,t,\alpha^*,\beta^*}$ which is now generated as $\mathsf{OT}_2(\sigma_{t,\alpha^*,\beta^*}, \mathsf{ots}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t}, \mathsf{lab}^{i,t+1}_{h,\mathsf{Z}_t})$ instead of $\mathsf{OT}_2(\sigma_{t,\alpha^*,\beta^*}, \mathsf{ots}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$.

  Indistinguishability between hybrids $\mathcal{H}_{t,2}$ and $\mathcal{H}_{t,3}$ follows directly from the sender's security of underlying malicious oblivious transfer protocol. Finally, observe that $\mathcal{H}_{t,3}$ is the same as hybrid $\mathcal{H}_t$.

■

6.4 Extensions

As in the semi-honest case, we discuss several extensions to the construction of two-round maliciously secure MPC.

Fairness. Assuming honest majority we obtain fairness in three rounds using techniques from [GLS15]. Specifically, we can change the function description to output a $n/2$-out-of-n secret sharing of the output. In the last round, the parties exchange their shares to reconstruct the output. Note that since the corrupted parties is in minority, it cannot learn the output of the function even if it obtains the second round messages from all the parties. Note that Gordon et al. [GLS15] showed that three rounds are necessary to achieve fairness. Thus this is optimal.

Semi-malicious security in Plain Model. We note that a simple modification of our construction in Figure 3 can be made semi-maliciously secure in the plain model. The modification is to use a two-round OT secure against semi-malicious receiver and semi-honest sender (e.g.,[NP01]) and achieve equivocability by sending two OT_1 messages in the first round having the same receiver's choice bit. Note that this is trivially equivocal since a simulator can use different choice bits in the OT_1 message. On the other hand, since a semi-malicious party is required to follow the protocol, it will always use the same choice bit in both the OT_1 messages.

References

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[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*, pages 260–274, 2005.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 119–135, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.

[AJL$^+$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 David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[BCL$^+$05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 361–377, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112, Chicago, IL, USA, May 2–4, 1988. ACM Press.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193, Paris, France, May 8–12, 2017. Springer, Heidelberg, Germany.

[BH15] Mihir Bellare and Viet Tung Hoang. Adaptive witness encryption and asymmetric password-based cryptography. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 308–331, Gaithersburg, MD, USA, March 30 – April 1, 2015. Springer, Heidelberg, Germany.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 784–796, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 190–213, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

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

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

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

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

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols, 2005. Version of December 2005. Available at `http://eccc.uni-trier.de/eccc-reports/2001/TR01-016`.

[CCM98] Christian Cachin, Claude Crépeau, and Julien Marcil. Oblivious transfer with a memory-bounded receiver. In *39th FOCS*, pages 493–502, Palo Alto, CA, USA, November 8–11, 1998. IEEE Computer Society Press.

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 33–65, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *51st FOCS*, pages 541–550, Las Vegas, NV, USA, October 23–26, 2010. IEEE Computer Society Press.

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 630–656, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[DG17] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 537–569, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[DHRS04] Yan Zong Ding, Danny Harnik, Alon Rosen, and Ronen Shaltiel. Constant-round oblivious transfer in the bounded storage model. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 446–472, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317, St. Louis, Missouri, October 22–24, 1990. IEEE Computer Society Press.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 74–94, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany.

[GGMP16] Sanjam Garg, Divya Gupta, Peihan Miao, and Omkant Pandey. Secure multiparty RAM computation in constant rounds. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 491–520, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GHL$^+$14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 405–422, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[GKK$^+$12] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 513–524, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled RAM. In Venkatesan Guruswami, editor, *56th FOCS*, pages 210–229, Berkeley, CA, USA, October 17–20, 2015. IEEE Computer Society Press.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled RAM from one-way functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 449–458, Portland, OR, USA, June 14–17, 2015. ACM Press.

[GLS15] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 63–82, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

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

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

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

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[GOVW12] Sanjam Garg, Rafail Ostrovsky, Ivan Visconti, and Akshay Wadia. Resettable statistical zero knowledge. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 494–511, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.

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

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

[HY16] Carmit Hazay and Avishay Yanai. Constant-round maliciously secure two-party computation in the RAM model. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 521–553, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[Jou04] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, September 2004.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31, Chicago, IL, USA, May 2–4, 1988. ACM Press.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 719–734, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

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

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457, Washington, DC, USA, January 7–9, 2001. ACM-SIAM.

[OS97] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *29th ACM STOC*, pages 294–303, El Paso, TX, USA, May 4–6, 1997. ACM Press.

[PS16] Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 217–238, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In S. Jajodia and P. Samarati, editors, *ACM CCS 00*, pages 245–254, Athens, Greece, November 1–4, 2000. ACM Press.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

A Our Model

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

A.1 The basic model of execution

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

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

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

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].)

A.2 Security of protocols

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

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

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

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

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

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

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

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

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

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

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

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

A.3 Hybrid protocols

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

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

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

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

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

The universal composition theorem. Let $\mathcal{F}$ be an ideal functionality. In its general form, the composition theorem basically says that if Π is a protocol that UC-realizes $\mathcal{F}$ then, for any $\mathcal{F}$-hybrid protocol ρ, we have that an execution of the composed protocol ρ^{Π} "emulates" an execution of protocol ρ. That is, for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and protocol ρ^{Π} or with $\mathcal{S}$ and protocol ρ, in a UC interaction. As a corollary, we get that if protocol ρ UC-realizes $\mathcal{F}$, then so does protocol ρ^{Π}. [5]

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

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

A.4 The Common Reference/Random String Functionality

In the common reference string (CRS) model [CF01, CLOS02], all parties in the system obtain from a trusted party a reference string, which is sampled according to a pre-specified distribution D. The reference string is referred to as the *CRS*. In the UC framework, this is modeled by an ideal functionality $\mathcal{F}_{CRS}^{D}$ 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 4.

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 $\mathcal{F}_{CRS}$.

[5] 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 some party send $(\texttt{crs}, sid, \rho)$ to that party.

Figure 4: The Common Reference String Functionality.

A.5 General Functionality

We consider the general-UC functionality $\mathcal{F}$, which securely evaluates any polynomial-time (possibly randomize) function $f : (\{0,1\}^{\ell_{in}})^n \to (\{0,1\}^{\ell_{out}})^n$. The functionality $\mathcal{F}_f$ is parameterized with a function f and is described in Figure 5. 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 $(\mathsf{generateOutput}, \mathsf{sid}, \mathcal{P}, P_i)$ from $\mathcal{S}$ the ideal functionality outputs $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, y_i)$ to P_i. (And ignores the message if inputs from all parties in $\mathcal{P}$ have not been received.)

Figure 5: General Functionality.

B Equivocal Receiver's Security in Oblivious Transfer

Using standard techniques for the literature (e.g. [CLOS02]) it is possible to add equivocal receiver's security to any OT protocol. We sketch a construction in below for the sake of completeness.

Lemma B.1 *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 $(\mathsf{ots}^0_{1,i}, \omega^0_i) \leftarrow \mathsf{OT}'_1(\sigma', x_i)$.
  3. For each $i \in [\lambda]$, prepare $(\mathsf{ots}^1_{1,i}, \omega^1_i) \leftarrow \mathsf{OT}'_1(\sigma', x_i)$.
  4. Output $\mathsf{ots}_1 := (y, \{\mathsf{ots}^0_{1,i}, \mathsf{ots}^1_{1,i}\}_{i\in[\lambda]})$ and $\omega := \left(\beta, \{\omega^0_i\}_{i\in[\lambda]}\right)$ if $\beta = 0$ and $\omega := \left(\beta, \left\{\omega^1_i\right\}_{i\in[\lambda]}\right)$ otherwise.
- $\mathsf{OT}_2(\sigma = (\sigma', r), \mathsf{ots}_1 = (y, \{\mathsf{ots}^0_{1,i}, \mathsf{ots}^1_{1,i}\}_{i\in[\lambda]}), (s_0, s_1))$: 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 $(\widetilde{C}^0, \{\mathsf{lab}^0_{i,b}\}_{i\in[\lambda],b\in\{0,1\}}) \leftarrow \mathsf{Garble}(1^\lambda, C_{y,s_0})$.
  2. Obtain $(\widetilde{C}^1, \{\mathsf{lab}^1_{i,b}\}_{i\in[\lambda],b\in\{0,1\}}) \leftarrow \mathsf{Garble}(1^\lambda, C_{r\oplus y,s_1})$.
  3. For each $i \in [\lambda]$, obtain $\mathsf{ots}^0_{2,i} \leftarrow \mathsf{OT}'_2(\sigma', \mathsf{ots}^0_{1,i}, (\mathsf{lab}^0_{i,0}, \mathsf{lab}^0_{i,1}))$.
  4. For each $i \in [\lambda]$, obtain $\mathsf{ots}^1_{2,i} \leftarrow \mathsf{OT}'_2(\sigma', \mathsf{ots}^1_{1,i}, (\mathsf{lab}^1_{i,0}, \mathsf{lab}^1_{i,1}))$.
  5. Output $\mathsf{ots}_2 := (\widetilde{C}^0, \widetilde{C}^1, \{\mathsf{ots}^0_{2,i}, \mathsf{ots}^1_{2,i}\}_{i\in[\lambda]})$.
- $\mathsf{OT}_3\left(\sigma = (\sigma', r), \mathsf{ots}_2 = (\widetilde{C}^0, \widetilde{C}^1, \{\mathsf{ots}^0_{2,i}, \mathsf{ots}^1_{2,i}\}_{i\in[\lambda]}), \omega = \left(\beta, \left\{\omega^\beta_i\right\}_{i\in[\lambda]}\right)\right)$: Compute
  1. For each $i \in [\lambda]$, recover $\mathsf{lab}_i := \mathsf{OT}'_3(\sigma', \mathsf{ots}^\beta_{2,i}, \omega^\beta_i)$.
  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.
- $\mathsf{Ext}_2\left(\tau, \mathsf{ots}_1 = \left(y, \left\{\mathsf{ots}^0_{1,i}, \mathsf{ots}^1_{1,i}\right\}_{i\in[\lambda]}\right)\right)$ proceeds as follows: For each $i \in [n]$, obtain $x_{0,i} := \mathsf{Ext}'_2(\tau, \mathsf{ots}_{1,i})$. 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 $\mathrm{IND}^{\mathrm{REAL}}_{\mathcal{A}}(1^\lambda, K_0, K_1)$ and $\mathrm{IND}^{\mathrm{IDEAL}}_{\mathcal{A}}(1^\lambda, K_0, K_1)$ are computationally indistinguishable. We argue this via the following sequence of hybrids.

- $\mathcal{H}_0$: This hybrid is the same as $\mathrm{IND}^{\mathrm{REAL}}_{\mathcal{A}}(1^\lambda, K_0, K_1)$.

- $\mathcal{H}_1$: 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 $\{\mathsf{ots}^0_{1,i}\}_{i\in[\lambda]}$ and $\{\mathsf{ots}^1_{1,i}\}_{i\in[\lambda]}$, 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, by a counting argument we make the claim that for any x_0, x_1 over the random choices of y we have that $\Pr[g(x_0) = y \wedge g(x_1) = r \oplus y]$ 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 $\mathsf{ots}^{1-\beta}_{2,i}$ are generated for each $i \in [\lambda]$. More specifically, for each $i \in [\lambda]$, we generate $\mathsf{ots}^{1-\beta}_{2,i} \leftarrow \mathsf{OT}_2\left(\sigma', \mathsf{ots}^{1-\beta}_{1,i}, (\mathsf{lab}^{1-\beta}_{i,x_{1-\beta,i}}, \mathsf{lab}^{1-\beta}_{i,x_{1-\beta,i}})\right)$.

  Indistinguishability between $\mathcal{H}_1$ and $\mathcal{H}_2$ can be reduced to the receiver'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. Sample $x \leftarrow \{0,1\}^\lambda$. Set $y := g(x_0)$.
3. For each $i \in [\lambda]$, prepare $(\mathsf{ots}^0_{1,i}, \omega^0_i) \leftarrow \mathsf{OT}'_1(\sigma', x_{0,i})$.
4. For each $i \in [\lambda]$, prepare $(\mathsf{ots}^1_{1,i}, \omega^1_i) \leftarrow \mathsf{OT}'_1(\sigma', x_{1,i})$.
5. Output $\left(\sigma := (\sigma', r), \mathsf{ots}_1 := (y, \{\mathsf{ots}^0_{1,i}, \mathsf{ots}^1_{1,i}\}_{i\in[\lambda]}), \omega_0 := \left(\beta, \{\omega^0_i\}_{i\in[\lambda]}\right), \omega_1 := \left(\beta, \{\omega^1_i\}_{i\in[\lambda]}\right)\right)$.

We are left to argue that for each β, the distribution $(\sigma, \mathsf{ots}_1, \omega_\beta)$ 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 $(\mathsf{ots}_1 = (y, \{\mathsf{ots}^0_{1,i}, \mathsf{ots}^1_{1,i}\}_{i\in[\lambda]}), \omega_\beta) \leftarrow \mathsf{OT}_1(\sigma, \beta)$.

- $\mathcal{H}_1$: 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 ots_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 $\mathsf{ots}^1_{1,i}$ values are generated. Specifically, for each $i \in [\lambda]$, we set $(\mathsf{ots}^1_{1,i}, \omega^1_i) \leftarrow \mathsf{OT}'_1(\sigma', x'_i)$ instead of $(\mathsf{ots}^1_{1,i}, \omega^1_i) \leftarrow \mathsf{OT}'_1(\sigma', x_i)$. 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. ■

C Completing the Proof of Lemma 5.2

Claim C.1 *Assuming the security of garbling scheme for circuits, $\mathcal{H}_{t-1} \approx_c \mathcal{H}_{t,1}$.*

Proof Let $\mathcal{A}$ be an adversary corrupting the set of parties $[n] \setminus H$ that can distinguish between $\mathcal{H}_t$ and $\mathcal{H}_{t,1}$ with non-negligible probability. We construct an adversary $\mathcal{B}$ breaking the security of garbling scheme.

$\mathcal{B}$ chooses an uniform random tape for every $j \notin H$ and interacts with the adversary $\mathcal{A}$. $\mathcal{B}$ computes the first round message $z_i, \{\mathsf{ots}_{1,t,\alpha,\beta}\}_{t \in A_i, \alpha,\beta \in \{0,1\}}$ for every $i \in H$ as in $\mathcal{H}_{t-1}$. $\mathcal{B}$ runs in its "head" a faithful execution of the protocol Φ using both honest and corrupted parties inputs. This yields the protocol transcript Z and the shared local state st^*. For every $i \in H$, it generates the second round message as follows:

1. 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 $b \in \{0,1\}$ $\mathsf{lab}^{i,T+1}_{k,b} := 0^\lambda$.
2. **for** each w from T down to $t+1$,
   (a) Parse ϕ_w as (i^*, f, g, h).
   (b) If $i = i^*$ then compute (where P is described in Figure 2)
   $$\left(\widetilde{\mathsf{P}}^{i,w}, \overline{\mathsf{lab}}^{i,w}\right) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{P}[i, \phi_w, v_i, \{\omega_{w,\alpha,\beta}\}_{\alpha,\beta}, \perp, \overline{\mathsf{lab}}^{i,w+1}]).$$
   (c) If $i \neq i^*$ then for every $\alpha, \beta \in \{0,1\}$, set $\mathsf{ots}^i_{2,w,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{ots}_{1,w,\alpha,\beta}, \mathsf{lab}^{i,w+1}_{h,0}, \mathsf{lab}^{i,w+1}_{h,1})$ and compute
   $$\left(\widetilde{\mathsf{P}}^{i,w}, \overline{\mathsf{lab}}^{i,w}\right) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{P}[i, \phi_w, v_i, \perp, \{\mathsf{ots}^i_{2,w,\alpha,\beta}\}_{\alpha,\beta}, \overline{\mathsf{lab}}^{i,w+1}]).$$
3. For every $i \in H$, let st^t_i be the secret local state of party P_i before the beginning of the t^{th} round of the computation phase. Interact with the garbled circuits challenger and give st^t_i as the challenge input and $\mathsf{P}[i, \phi_t, v_i, \{\omega_{t,\alpha,\beta}\}_{\alpha,\beta}, \perp, \overline{\mathsf{lab}}^{i,t+1}]$ as the challenge circuit if $i = i^*$ and $\mathsf{P}[i, \phi_t, v_i, \perp, \{\mathsf{ots}^i_{2,t,\alpha,\beta}\}_{\alpha,\beta}, \overline{\mathsf{lab}}^{i,t+1}]$ as the challenge circuit if $i \neq i^*$ where $\mathsf{ots}^i_{2,t,\alpha,\beta} \leftarrow \mathsf{OT}_2(\mathsf{ots}_{1,t,\alpha,\beta}, \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{lab}^{i,t+1}_{h,1})$. Obtain $\widetilde{\mathsf{P}}^{i,t}$ and $\{\mathsf{lab}^{i,t}_k\}_{k \in [\ell]}$.
4. **for** each w from $t-1$ down to 1:
   (a) Parse ϕ_w as (i^*, f, g, h).
   (b) Set $\alpha^* := \mathsf{st}^*_f$, $\beta^* = \mathsf{st}^*_g$ and $\gamma^* := \mathsf{st}^*_h$.

(c) If $i = i^*$, compute

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

(d) Else, compute $\mathsf{ots}^i_{2,w,\alpha^*,\beta^*}$ as $\mathsf{OT}_2(\mathsf{ots}_{1,t,\alpha^*,\beta^*}, \mathsf{lab}_h^{i,w+1}, \mathsf{lab}_h^{i,w+1})$ and generate

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

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

Notice that if the garbling $\widetilde{\mathsf{P}}^{i,t}$ is generated using the honest procedure then the messages sent to $\mathcal{A}$ are distributed identically to $\mathcal{H}_{t-1}$. Else, they are distributed identically to $\mathcal{H}_{t,1}$. Thus, $\mathcal{B}$ breaks the security of garbling scheme for circuits which is a contradiction. ■

Claim C.2 *Assuming the receiver security of oblivious transfer, we have $\mathcal{H}_{t,1} \approx_c \mathcal{H}_{t,2}$.*

Proof Let $\mathcal{A}$ be an adversary corrupting the set of parties $[n] \setminus H$ that can distinguish between $\mathcal{H}_{t,1}$ and $\mathcal{H}_{t,2}$ with non-negligible probability. We construct an adversary $\mathcal{B}$ breaking the receiver security of oblivious transfer.

$\mathcal{B}$ chooses an uniform random tape for every $j \notin H$ and interacts with the adversary $\mathcal{A}$. For every $i \in H$, $\mathcal{B}$ generates z_i and $\mathsf{ots}_{1,w,\alpha,\beta}$ for every $w \in \cup_{i\in H} A_i \setminus \{t\}$ as in $\mathcal{H}_{t,1}$. $\mathcal{B}$ runs in its "head" a faithful execution of the protocol Φ using both honest and corrupted parties inputs. This yields the protocol transcript Z and the shared local state st^*. Set $\alpha^* := \mathsf{st}^*_f$, $\beta^* = \mathsf{st}^*_g$ and $\gamma^* := \mathsf{st}^*_h$. Let $\phi_t = (i^*, f, g, h)$. For $(\alpha, \beta) \neq (\alpha^*, \beta^*)$, interact with the OT challenger and send Z_t, $v_{i,h} \oplus \mathsf{NAND}(v_{i,f} \oplus \alpha, v_{i,g} \oplus \beta)$ as the challenge bits. Obtain $\mathsf{ots}_{1,t,\alpha,\beta}$ as the challenge first OT message. Sample $\omega_{t,\alpha^*,\beta^*}$ uniformly at random and compute $\mathsf{ots}_{1,t,\alpha^*,\beta^*}$ as $\mathsf{OT}_1(1^\lambda, \mathsf{Z}_t; \omega_{t,\alpha^*,\beta^*})$. For each $i \in H$, send $z_i, \{\mathsf{ots1}, w, \alpha, \beta\}_{w\in A_i,\alpha,\beta\in\{0,1\}}$ on behalf of the honest party. Generate the second round message as in $\mathcal{H}_{t,1}$.

Notice that if challenge bit is Z_t then the distribution of messages to $\mathcal{A}$ is identical to $\mathcal{H}_{t,2}$. Else, it is identical to $\mathcal{H}_{t,1}$. Thus, $\mathcal{B}$ breaks the receiver security of oblivious transfer. ■