Two-Round Multiparty Secure Computation Minimizing Public Key Operations [*]

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

Peihan Miao
University of California, Berkeley
peihan@berkeley.edu

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

Abstract

We show new constructions of semi-honest and malicious two-round multiparty secure computation protocols using only (a fixed) $\mathsf{poly}(n, \lambda)$ invocations of a two-round oblivious transfer protocol (which use expensive public-key operations) and $\mathsf{poly}(\lambda, |C|)$ cheaper one-way function calls, where λ is the security parameter, n is the number of parties, and C is the circuit being computed. All previously known two-round multiparty secure computation protocols required $\mathsf{poly}(\lambda, |C|)$ expensive public-key operations.

1 Introduction

Secure multiparty computation (MPC) allows a set of mutually distrusting parties to compute a joint function on their private inputs with the guarantee that only the output of the function is revealed and everything else about the private inputs of the parties is hidden. This is a classic problem in cryptography and was originally studied by Yao [Yao82] for the case of two parties. Later, Goldreich, Micali and Wigderson [GMW87] considered the multiparty case and gave protocols for securely computing any multiparty functionality.

A key metric in determining the efficiency of a secure computation protocol is its *round complexity* or in other words, the number of sequential messages exchanged between the parties. Starting with the first constant round protocol by Beaver, Micali and Rogaway [BMR90], there has been a tremendous amount of research to reduce the round complexity to its *absolute minimum.* It was shown in [HLP11] that two rounds are necessary to securely compute certain functionalities and a sequence of works have tried to realize this goal. The first two-round construction was obtained by Garg, Gentry, Halevi and Raykova based on indistinguishability obfuscation [GGHR14, GGH$^+$13]. Subsequently, a sequence of works improved the needed assumptions, first to witness encryption [GLS15, GGSW13], and then to learning with errors assumption [MW16, BP16, PS16]. Improving these results, recent works obtained two-round constructions based on the DDH assumption

[*] Research supported in part from 2017 AFOSR YIP Award, DARPA/ARL SAFEWARE Award W911NF15C0210, AFOSR Award FA9550-15-1-0274, NSF CRII Award 1464397, 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.

[BGI16, BGI17] (for the case of constant number of parties) or on bilinear maps [GS17] (in the general case). Finally, very recent results have also yielded constructions based on the minimal assumption of two-round oblivious transfer [BL18, GS18].

Apart from round complexity, another metric that is crucial for computational efficiency in MPC protocols is the *number of public-key operations* performed by each party. Typically, public key operations are orders of magnitude more expensive than symmetric key operations and minimizing them typically leads to more efficient protocols. The question of minimizing public key operations in secure computation was first considered by Beaver [Bea96] for the case of oblivious transfer. In particular, Beaver gave a construction for obtaining a large number $L \gg \lambda$ of oblivious transfers (OTs) using only a fixed number λ public key operations along with the use of $\mathsf{poly}(L)$ cheaper one-way function calls. This task of extending λ OTs to a larger L OTs using only one-way functions is referred to as oblivious transfer extension. Following Beaver's result, a rich line of work [IKNP03, Nie07, HIKN08, KK13] gave concretely efficient protocols for OT extension which have served as a crucial ingredient in the design of several concretely efficient secure computation protocols [HIK07, NNOB12, ALSZ17, KRS16].

In this work, we are interested in getting the best of both worlds, namely, constructing two-round MPC protocols while minimizing the number of public-key operations performed. Indeed, the number of public-key operations in the prior two-round MPC protocols grows with the size of the circuit computed. Given this state of affairs, we would like to address the following question.

Can we construct two-round, secure multiparty computation protocols where the number of public key operations performed by each party is independent of the size of the circuit being computed?

1.1 Our Results

We give a positive answer to the above question. We show new constructions of semi-honest and malicious two-round, multiparty computation protocols where the number of public key operations performed by each party is a *fixed polynomial* (in the security parameter and the number of participants) and is *independent* of the circuit size of the function being computed. Further, we prove the security of these protocols under the *minimal assumption* that two-round semi-honest/malicious oblivious transfer (OT) exists. More formally, our main theorem is:

Theorem 1.1 *Let $\mathcal{X} \in \{$semi-honest in plain model, malicious in common random/reference sting model$\}$. Assuming the existence of a two-round $\mathcal{X}$ secure OT protocol, there exists a two-round, $\mathcal{X}$ secure, n-party protocol computing a function f (represented as a circuit C_f) where the number of public key operations performed by each party is $\mathsf{poly}(n, \lambda)$. Here, $\mathsf{poly}(\cdot)$ is a fixed polynomial independent of $|C_f|$ and λ is the security parameter.*

The focus of this work is theoretical feasibility rather than concrete optimization of the polynomial. We leave the goal of obtaining concretely efficient protocols for future work. Additionally, in the malicious case, this work focuses on obtaining protocols in the common random/reference string model. Obtaining round optimal MPC protocols in the plain model [GMPP16, ACJ17, BHP17, COSV17, HHPV17, BGJ$^+$17, BL18] has been a problem of significant interest and we expect that our techniques will be useful in reducing the number of public-key operations needed in these protocols. We leave this as an open problem.

2 Technical Overview

In this section, we give a high-level overview of the main challenges and the techniques used to overcome them in our construction of two-round MPC protocols minimizing the number of public key operations.

Starting Point. The starting point of our work is the recent results of Benhamouda and Lin [BL18] and Garg and Srinivasan [GS18] that provide constructions of two-round, secure multiparty computation (MPC) protocol based on two-round oblivious transfer. These works provide a method of squishing the round complexity of an arbitrary round secure computation protocol to just two rounds. The key idea behind this method is the concept of "talking garbled circuits," i.e., garbled circuits that can interact with each other by sending and receiving messages. Let us briefly explain how this primitive helps in squishing the round complexity of a multi-round MPC protocol.

To squish the round complexity, each party generates "talking garbled circuits" that emulates its actions as per the specification of the multi-round MPC protocol. The parties then broadcast these "talking garbled circuits" so that every party has access to the "talking garbled circuits" of every other party. Finally, all parties evaluate these "talking garbled circuits" that internally executes the multi-round MPC protocol. This step does not involve any further interactions between the parties. Thus, the only overhead in the round complexity of this approach is the number of rounds needed for generating the "talking garbled circuits."

Let us give a very high level overview of how the "talking garbled circuits" are generated. In these two works, the "talking garbled circuits" are generated via a two-round protocol that makes use of (plain) garbled circuits and two-round oblivious transfer (OT).[1] At the end of the two rounds, every party has access to every other party's "talking garbled circuits" and can evaluate them without any further interaction. The first round of this two-round protocol can be visualized as setting up a channel for the garbled circuits to communicate. Without going into the actual details on how this is achieved, we note that this step involves generating several first round OT messages. Next, in the second round, the actual garbled circuits are sent which interact with each other via the channel set up in the first round. Again, without going into the details, a message sent from one party (the sender) to another party (the receiver) is communicated via the sender's garbled circuit outputting the randomness used in generating a subset of the first round OT messages and the receiver's garbled circuit outputting some second round OT messages.

Computational Overhead. One major source of inefficiency in the approaches of [BL18, GS18] is the number of expensive OT instances needed. In particular, these protocols use $\Omega(1)$ OTs in enabling the garbled circuits to communicate a single bit. Hence, the number of OTs needed for compiling an arbitrary secure computation protocol grows with the circuit size of the function being computed.[2] Our goal is to remove this dependency between the number of OTs needed and the circuit size of the function being computed.

Can we use OT extension? A natural first attempt to minimize the number of instances of oblivious transfer would to be use an OT extension protocol [Bea96, IKNP03]. We need this OT

[1] Recall that in a two-round oblivious transfer, the first message is generated by the receiver and it encodes the receiver's choice bit and the second message is generated by the sender and it encodes its two messages.

[2] In fact, the number of OTs grows with the computational complexity of the underlying multiparty protocol.

extension protocol to run in two-rounds, as otherwise the protocol for computing "talking garbled circuits" will run in more rounds. Further, we need the OT extension protocol to satisfy the following three properties for it to be useful in constructing "talking garbled circuits." We also explain why a general two-round OT satisfies each of these properties.

1. **Delegatability.** For every OT computed between a sender and a receiver, the receiver should be able to delegate its decryption capabilities for that OT to any party by revealing a decryption key. This key and the transcript could then be used to compute the message that the receiver would have obtained in the OT execution. A general two-round OT satisfies delegatability as revealing the receiver's random coins allows any party to obtain the receiver's message.

2. **Independence.** We require independence between multiple parallel invocations of the underlying OT protocol. More specifically, revealing the receiver's delegation key for one of the instances of an OT execution does not affect the receiver security for the other OTs. Again, a general two-round OT satisfies independence as each OT instance is generated using an independent random tape.

3. **Availability of Delegation Keys.** The keys for delegating the decryption must be available at the end of the first round i.e., after the receiver sends its message. This property is trivially satisfied by a two-round OT as the delegation key is in fact the receiver's random tape.

Let us first explain the intuition on why these three properties are required for the construction of "talking garbled circuits." The delegatability property is required since the garbled circuits sent in the second round reveal the delegation keys for a subset of the OT messages generated in the first round. Recall that this is required for one garbled circuit to send a message to another. The key availability property is needed since the delegation keys are to be hardwired in the second round garbled circuits so that the appropriate delegation keys can be output by these circuits during evaluation. The independence property is needed since the second round garbled circuits reveal the delegation keys for only a subset of the first round OT messages. We need the other OT messages to still be secure.

We stress that even though the above three properties are trivially satisfied by every two-round OT, a two-round OT extension protocol need not satisfy all of them. To demonstrate this, let us first see why does the two-round version of Beaver's OT extension protocol [Bea96, GMMM17] not satisfy all the properties.

Why doesn't Beaver's OT extension work? In order to understand why this does not work, we first recall a two-round version [GMMM17] of the OT extension protocol of Beaver that expands λ two-round, base OTs to $L = \mathsf{poly}(\lambda)$ OTs. In the first round of the OT extension protocol, the receiver (having input $c \in \{0,1\}^L$) samples a "short" seed s of a $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^L$ and computes $e = c \oplus \mathsf{PRG}(s)$. Additionally, it computes λ first round OT messages using s as its choice bits. It sends these OT messages along with e to the sender. The sender garbles a circuit C that has its messages $\{\mathsf{msg}_{i,0}, \mathsf{msg}_{i,1}\}_{i\in[L]}$ hardwired along with the string e received in the first round. The circuit C takes as input the λ-bit string s, expands it to L bits using the PRG and uses it to unmask e to obtain c. Specifically, it computes $c := e \oplus \mathsf{PRG}(s)$, and outputs $\{\mathsf{msg}_{i,c[i]}\}_{i\in[L]}$. The sender sends this garbled circuit and uses the λ second round OT messages to communicate the labels of the garbled circuit to the receiver. The receiver decrypts the labels corresponding to the bits of its seed s and uses it to evaluate the garbled circuit to obtain $\{\mathsf{msg}_{i,c[i]}\}_{i\in[L]}$.

The above OT extension protocol of Beaver is delegatable as revealing all the randomness used by the receiver allows any party to decrypt all the messages. However, the protocol does not satisfy the independence requirement as the randomness used for generating L different OTs is highly correlated. In fact, revealing all the random coins for generating the first round OT messages compromises the security of all the L OTs.

Delegatable and Independent Two-Round OT Extension. Towards constructing an OT extension that satisfies all the properties, we first construct a protocol that is both delegatable and independent. In the new protocol, the receiver's first round message is the same as before. However, the sender's message is generated differently. In particular, the sender samples a set of masks $M = \{\mathsf{m}_{i,0}, \mathsf{m}_{i,1}\}_{i\in[L]}$ where each mask $\mathsf{m}_{i,b}$ is a random string with the same length as $\mathsf{msg}_{i,b}$. It constructs the circuit C (described above) with the set of masks hardwired in place of the messages. It garbles this circuit. It additionally computes $ct_{i,b} = \mathsf{msg}_{i,b} \oplus \mathsf{m}_{i,b}$ for each $i \in [L]$ and $b \in \{0,1\}$ and sends the garbled circuit, the set $\{ct_{i,b}\}_{i\in[L],b\in\{0,1\}}$ and λ second round OT messages to communicate the labels of the garbled circuit to the receiver. The receiver then recovers the labels corresponding to its seed s, evaluates the garbled circuit to obtain $\{\mathsf{m}_{i,c[i]}\}_{i\in[L]}$, and computes $\mathsf{msg}_{i,c[i]} = ct_{i,c[i]} \oplus \mathsf{m}_{i,c[i]}$ for every $i \in [L]$.

This scheme is delegatable as the receiver can use $\mathsf{m}_{i,c[i]}$ as the delegation key. It is also independent, as revealing $\mathsf{m}_{i,c[i]}$ does not leak any information of $c[k]$ for $k \neq i$. However, this construction does not satisfy the third property, namely key availability. This is because $\mathsf{m}_{i,c[i]}$ can be computed by the receiver only at the end of the second round and is not available at the end of the first round.

Weakening the Key Availability Property. We first observe that we can in fact, weaken the key availability property. Recall that the key availability property requires the delegation keys to be available at the end of the first round so that they can be hardwired inside the garbled circuits that performs the communication. However, for the construction to work, we just need the delegation keys to be given as inputs to these garbled circuits and need not be hardwired. We will now construct a two-round, OT extension that satisfies the weakened key availability property. For the ease of exposition, let us overload the notation and call the these communicating garbled circuits (sent in the second round) as "talking garbled circuits."

Satisfying All Properties. Recall that the problem with the previous approach was because the receiver could evaluate the sender's garbled circuit only at the end of the second round. Our solution to the key availability problem is in having the receiver "offload" its evaluation of this garbled circuit. This solution makes use of the fact that in the MPC setting the sender and the receiver are connected via a *simultaneous message exchange* model. At a high level, we require the sender to send its garbled circuit in the first round. The receiver now garbles a wrap-circuit, which has the sender's garbled circuit hardwired in it. This wrap-circuit evaluates the sender's circuit inside and translates its output to the labels of the "talking garbled circuits." In particular, the receiver "offloads" the evaluation of the sender's garbled circuit via the wrap-circuit which helps in achieving the weakened key availability property. Let us explain our idea in more detail.

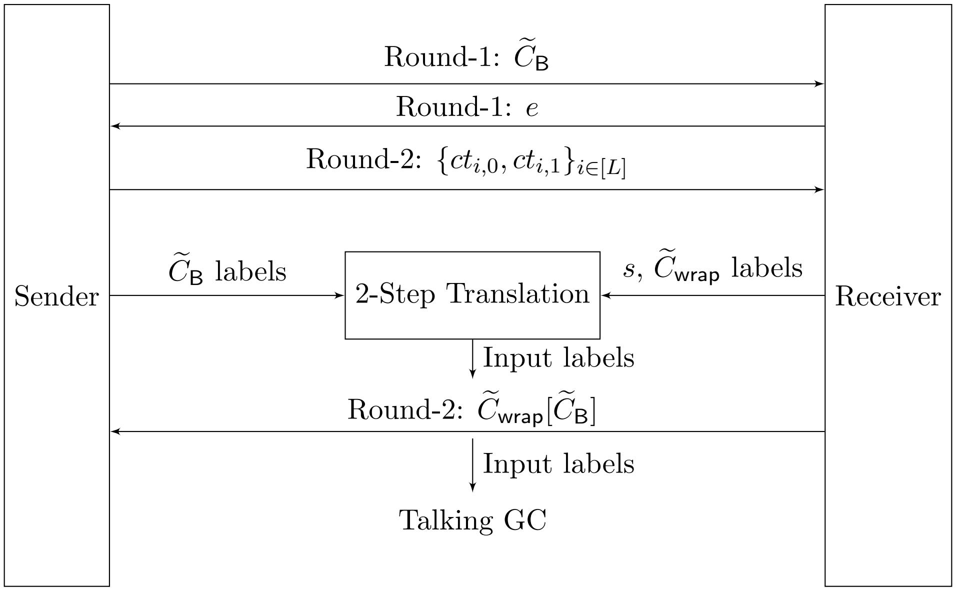
Round-1: $\widetilde{C}_{\mathsf{B}}$
Round-1: e
Round-2: $\{ct_{i,0}, ct_{i,1}\}_{i\in[L]}$
$\widetilde{C}_{\mathsf{B}}$ labels
2-Step Translation
s, $\widetilde{C}_{\mathsf{wrap}}$ labels
Sender
Receiver
Input labels
Round-2: $\widetilde{C}_{\mathsf{wrap}}[\widetilde{C}_{\mathsf{B}}]$
Input labels
Talking GC

Figure 1: Semi-honest OT extension satisfying delegatability, independence and weakened key availability

Key Idea: "Offloading" Garbled Circuit Evaluation. We first give the description of the protocol and then explain why it satisfies all the three properties. The key steps in the protocol are depicted in Figure 1.

In the new protocol, the receiver's first round message is unchanged. Additionally, in the first round, the sender samples the random set M as before and constructs a circuit C_{B} that has the set M hardwired in it. This circuit takes as input a seed s, expands it using the PRG and outputs $\{\mathsf{m}_{i,\mathsf{PRG}(s)[i]}\}_{i\in[L]}$. The sender garbles C_{B} to obtain a garbled circuit $\widetilde{C}_{\mathsf{B}}$ and sends this to the receiver.

In the second round, the sender computes $ct_{i,0} = \mathsf{msg}_{i,0} \oplus \mathsf{m}_{i,e[i]}$ and $ct_{i,1} = \mathsf{msg}_{i,1} \oplus \mathsf{m}_{i,1-e[i]}$ (where e is obtained from the receiver's first round message) and sends $\{ct_{i,b}\}_{i\in[L],b\in\{0,1\}}$ to the receiver. The receiver constructs a wrap-circuit C_{wrap} that has $\widetilde{C}_{\mathsf{B}}$ and the input labels for the "talking garbled circuits" hardwired in it. C_{wrap} takes as input the labels for evaluating $\widetilde{C}_{\mathsf{B}}$, evaluates it using these labels to obtain $\{\mathsf{m}_{i,\mathsf{PRG}(s)[i]}\}_{i\in[L]}$, and outputs a set of labels corresponding to $\{\mathsf{m}_{i,\mathsf{PRG}(s)[i]}\}_{i\in[L]}$. The output will later be treated as the input labels for evaluating the "talking garbled circuits." The receiver garbles C_{wrap} and sends the garbled circuit $\widetilde{C}_{\mathsf{wrap}}$ to the sender.

Notice that $\mathsf{m}_{i,\mathsf{PRG}(s)[i]}$ can serve as the delegation keys as it can be used to unmask $ct_{i,c[i]}$ to obtain $\mathsf{msg}_{i,c[i]}$, and the other message $\mathsf{msg}_{i,1-c[i]}$ is hidden. This approach inherits the delegatability and independence from the previous approach. Now, this scheme also satisfies the weakened key availability property! In particular, the delegation keys are passed to the "talking garbled circuits" via the wrap circuit.

How to obtain labels for evaluating $\widetilde{C}_{\mathsf{wrap}}$? However, there is one question that we have not answered yet. In particular, how to obtain the labels for evaluating the garbled wrap-circuit $\widetilde{C}_{\mathsf{wrap}}$?

Recall that the warp-circuit C_{wrap} takes as input the labels for evaluating $\widetilde{C}_{\mathsf{B}}$. Hence, to evaluate $\widetilde{C}_{\mathsf{wrap}}$ we need its input labels that correspond to the labels for evaluating $\widetilde{C}_{\mathsf{B}}$. We therefore need a two-step translation mechanism: one from the seed s to the labels for evaluating $\widetilde{C}_{\mathsf{B}}$ and then from these labels to the labels for evaluating $\widetilde{C}_{\mathsf{wrap}}$.

For this purpose, we use the two-round MPC protocol from [BL18, GS18] to securely compute the two-step translation functionality. This functionality takes as input the seed s and the set of labels for $\widetilde{C}_{\mathsf{wrap}}$ from the receiver and the set of labels for $\widetilde{C}_{\mathsf{B}}$ from the sender. It first chooses the labels of $\widetilde{C}_{\mathsf{B}}$ that correspond to the string s. It then outputs the labels of $\widetilde{C}_{\mathsf{wrap}}$ that correspond to those chosen labels of $\widetilde{C}_{\mathsf{B}}$. Given such a two-round MPC protocol, we can run this protocol in parallel of the aforementioned protocol to obtain the labels for evaluating $\widetilde{C}_{\mathsf{wrap}}$. We then evaluate $\widetilde{C}_{\mathsf{wrap}}$ to obtain the labels for evaluating the "talking garbled circuits." Note that the circuit size computing this two-step translation functionality is polynomially dependent on λ and is independent of L and hence we can use these two-round MPC results to securely compute this functionality. This helps in minimizing the number of public key operations.

Tackling Malicious Adversaries. Plugging the above OT extension protocol into the compilers of [BL18, GS18] gives us the desired result in the semi-honest setting. However, a couple of major challenges arise in the malicious setting.

1. **Adaptive Security.** The first issue arises because a malicious receiver might wait until it receives the garbled circuit $\widetilde{C}_{\mathsf{B}}$ before choosing its seed s. This leads to adaptive security issues [BHR12] in garbling C_{B}.

2. **Input Dependent Abort.** The second issue arises because a malicious sender might generate an ill-formed $\widetilde{C}_{\mathsf{B}}$ that may lead to an honest receiver to abort on specific choices of the receiver's input. This leaks information about the receiver's input to the sender. To give a concrete example, a corrupted sender might generate $\widetilde{C}_{\mathsf{B}}$ such that it outputs $\perp$ if the first bit of $\mathsf{PRG}(s)$ is 1 instead of outputting the valid mask. Thus, if the honest receiver aborts then the sender can recover $c[1]$ from $e[1]$.

Solving these two issues requires development of new tools and techniques which we now elaborate.

Solving Adaptive Security Issue. A tempting approach to solving this issue is use the recent constructions of adaptively secure garbling [HJO$^+$16, JW16, JKK$^+$17] to generate $\widetilde{C}_{\mathsf{B}}$. However, this does not work! Recall that the length of the garbled input of an adaptively secure garbling scheme must at least grow with the output length of the circuit [AIKW13]. In our case, the output length of C_{B} is L, hence the garbled input of $\widetilde{C}_{\mathsf{B}}$ grows with L. Therefore, the circuit size of the two-step translation functionality that first translates the seed s to the garbled input of $\widetilde{C}_{\mathsf{B}}$ must grow with L. This implies that the number of public key operations in the two-round protocol that securely computes this functionality grows with L. This kills the efficiency of the overall protocol.

On the one hand, we need our garbling scheme to satisfy the stronger notion of adaptive security and on the other hand, we need to minimize the number of public key operations. These two requirements seem contradictory to each other and it seems that we need to trade one requirement in order to achieve the other. We resolve this deadlock by observing that full blown adaptive security is not needed in garbling C_{B}. We note that it is sufficient for this garbling scheme to be *somewhere adaptive*. Let us explain this in more detail.

To understand our approach, the first step is to break the circuit C_{B} down to L individual circuits $C_1, \ldots, C_L$ where C_i has $\{\mathsf{m}_{i,0}, \mathsf{m}_{i,1}\}$ hardwired and outputs $\mathsf{m}_{i,\mathsf{PRG}(s)[i]}$ on input s. The garbled circuit $\widetilde{C}_{\mathsf{B}}$ comprises of garbled versions of each C_i, i.e., $\widetilde{C}_1, \ldots, \widetilde{C}_L$. The key trick we employ in garbling $C_1, \ldots, C_L$ is that we use the *same set of input labels* in generating each $\widetilde{C}_i$. Notice that even though we break C_{B} down to L circuits, the garbled input for $\widetilde{C}_{\mathsf{B}}$ only grows with the input length of C_{B} and is independent of L. To simulate $\widetilde{C}_{\mathsf{B}}$, we design a sequence of carefully chosen hybrids where in each hybrid, it is sufficient to simulate a single $\widetilde{C}_i$. But things get complicated as the simulation of this $\widetilde{C}_i$ requires knowledge of the adaptively chosen s. It seems that we again run into the adaptive security issue. However, notice that the output length of the circuit C_i is independent of L and thus the length of the garbled input for $\widetilde{C}_i$ (and hence all other $\widetilde{C}_j$, $j \neq i$) need not grow with L! Thus, we can now use the standard tricks in the adaptive garbling circuits literature to "adaptively garble" C_i. We now explain how this is done.

Instead of sending the garbled circuits $\{\widetilde{C}_i\}_{i \in [L]}$ in the clear, we encrypt them using a somewhere equivocal encryption scheme [HJO$^+$16] and send the ciphertext as the garbled circuit $\widetilde{C}_{\mathsf{B}}$. The key for decrypting this ciphertext is revealed in the garbled input along with the labels for evaluating each $\widetilde{C}_i$. Recall that we use the same set of labels for evaluating each $\widetilde{C}_i$. Intuitively, a somewhere equivocal encryption allows to equivocate a bunch of positions of a ciphertext with arbitrary message values. What makes a somewhere equivocal encryption different from a fully equivocal encryption is that the size of the key only grows with the number of positions that are to be equivocated and is otherwise independent of the message size. Somewhere equivocal encryption allows us to solve the above adaptivity issue as we can equivocate the positions that correspond to $\widetilde{C}_i$ in the ciphertext to a simulated circuit (that can depend on the adaptively chosen s) by deriving a suitable key. Further, the size of the garbled input (that also includes the key) only grows with the size of $\widetilde{C}_i$ and is independent of L. This helps us in ensuring that the circuit size of the two-step translation functionality is independent of L.

Solving Input Dependent Aborts. Suppose the sender sends a proof that $\widetilde{C}_{\mathsf{B}}$ is correctly generated, then the problem of input dependent aborts does not arise. We additionally require this proof to be zero-knowledge so that it does not leak any information about the sender's secrets to the receiver. A natural approach would be to give a Non-Interactive Zero-Knowledge proof (NIZK). However, we only know constructions of NIZK based on public key assumptions such as trapdoor permutations or factoring. Furthermore, the number of public key operations in computing a NIZK proof grows with the instance size. Here, the instance size grows with the size of C_{B} which is at least L. This again kills the efficiency.

Our approach to solving this issue is to design a two-round, special purpose zero-knowledge proof (in the CRS model) where the number of public key operations is *independent* of the instance size. Indeed, given such a zero-knowledge proof, we can solve the problem of input dependent aborts and also ensure that the number of public key operations is independent of L. We now explain the main ideas behind this construction.

Let us first consider the simpler task of constructing a two-round, zero-knowledge proof with constant soundness error where the number of public key operations is independent of the instance size. We first observe that if we allow one more round of interaction then we know constructions (e.g., Blum's Hamiltonicity protocol) that completely avoid any public key operations. The main idea behind our construction is a method of compressing the round complexity of these protocols (in the simultaneous message exchange model) using a small number of public key operations (that is independent of the instance size). To explain the idea, let us take the example of compressing the Blum's Hamiltonicity protocol to two rounds using a two-round oblivious transfer. The Blum's protocol can be abstractly described using three messages: zk_1 sent by the prover in the first round, a random bit b sent by the verifier in the second round and $\mathsf{zk}_{3,b}$ sent by the prover in the third round.

To compress the protocol to two rounds, we require the verifier to send a receiver OT message with b as its choice bit in the first round. In addition to sending zk_1 in the first round, the prover also sends commitment (c_0, c_1) to $\mathsf{zk}_{3,0}$ and $\mathsf{zk}_{3,1}$ respectively. In the second round, the sender sends a sender OT message with the randomness used to compute c_0 and c_1 as its messages.[3] The receiver obtains the randomness used in generating c_b and then uses it to check if $(\mathsf{zk}_1, b, \mathsf{zk}_{3,b})$ is a valid proof. Note that to minimize the number of public key operations, the length of the random string used to generate the commitment should be independent of the size of the message. This is indeed true when we use a pseudorandom generator to expand the length of the randomness to any desired length.

The above idea helps us in achieving constant soundness error but to be useful in solving the problem of input dependent aborts, we need the protocol to have negligible soundness error. One approach to achieve negligible soundness is to do a parallel repetition of the constant soundness protocol but it is well-known that parallel repetition is not guaranteed to preserve the zero-knowledge property. Fiege and Shamir [FS90] showed that parallel repetition preserves the weaker property of witness indistinguishability and we make use of this fact to to achieve the stronger property of zero-knowledge. In our actual construction, we incorporate a trapdoor (such as pre-image of a one-way function) in the CRS and the simulator uses this trapdoor while generating the zero-knowledge proof. Witness indistinguishability guarantees that no verifier can distinguish between the prover's messages that uses the real witness and the simulator's messages that uses the trapdoor witness. This helps us achieve *zero-knowledge* against malicious verifiers and parallel repetition helps us achieve *negligible* soundness error against cheating provers. Additionally, the number of public key operations is a fixed polynomial in the security parameter and is independent of the instance size. We believe that this primitive may be of independent interest.

3 Preliminaries

We recall some standard cryptographic definitions in this section. Let λ denote the security parameter. A function $\mu(\cdot) : \mathbb{N} \rightarrow \mathbb{R}^+$ is said to be negligible if for any polynomial $\mathsf{poly}(\cdot)$ there exists 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.

For a binary string $x \in \{0,1\}^n$, we denote the i^{th} bit of x by $x[i]$. Similarly, we denote the substring of x from the i^{th} to j^{th} position for any $i \leq j$ by $x[i,j]$. For any $\overline{\mathsf{lab}} := \{\mathsf{lab}_{i,0}, \mathsf{lab}_{i,1}\}_{i \in [L]}$ where $\mathsf{lab}_{i,b} \in \{0,1\}^*$ and a string $c \in \{0,1\}^L$, we define $\mathsf{Projection}(c, \overline{\mathsf{lab}}) = \{\mathsf{lab}_{i,c[i]}\}_{i \in [L]}$. We treat the output of $\mathsf{Projection}$ as a string. That is, we treat the output as $\|_{i \in [L]}(\mathsf{lab}_{i,c[i]})$.

[3] We assume that given the randomness, we can obtain the message that is committed.

3.1 Selective Garbled Circuits

We recall the definition of selectively secure garbled circuits [Yao82] (see 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 $(\mathsf{Garble}, \mathsf{Eval})$. Very roughly, Garble is the circuit garbling procedure and Eval the corresponding evaluation procedure. We use a formulation where input labels for a garbled circuit are provided as input to the garbling procedure rather than generated as output. This simplifies the presentation of our construction. We additionally model security wherein the simulator is provided with a set of labels corresponding to the input. This helps in simplifying the security proofs. More formally:

- $\widetilde{\mathsf{C}} \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{C}, \{\mathsf{lab}_{w,b}\}_{w\in\mathsf{inp}(C),b\in\{0,1\}}\right)$: Garble takes as input a security parameter λ, a circuit C, and input labels $\mathsf{lab}_{w,b}$ where $w \in \mathsf{inp}(C)$ ($\mathsf{inp}(C)$ is the set of input wires to the circuit C) and $b \in \{0,1\}$. This procedure outputs a *garbled circuit* $\widetilde{\mathsf{C}}$. We assume that for each w, b, $\mathsf{lab}_{w,b}$ is chosen uniformly from $\{0,1\}^\lambda$.

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

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

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

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

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

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

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

3.2 Somewhere Adaptive Garbled Circuits

In this section, we define and construct somewhere adaptive garbled circuits. Intuitively, somewhere adaptive garbled circuits satisfy the stronger notion of adaptive security in the computation of a particular block of the output. Before we define this primitive, we give a notation to denote circuits.

Circuit Notation. We model a circuit $C : \{0,1\}^n \to \{0,1\}^{m\lambda}$ as a sequence of m circuits $C_1, C_2, \ldots, C_m$ where $C_i(x) = C(x)[(i-1)\lambda + 1, i\lambda]$ for every $x \in \{0,1\}^n$ and $i \in [m]$.

We now give the definition of somewhere adaptive garbled circuits.

Definition 3.1 *A* somewhere *adaptive garbling scheme for circuits is a tuple of PPT algorithms* $(\mathsf{SAdpGarbleCkt}, \mathsf{SAdpGarbleInp}, \mathsf{SAdpEvalCkt})$ *such that:*

- $(\widetilde{C}, \mathsf{state}) \leftarrow \mathsf{SAdpGarbleCkt}(1^\lambda, C)$: *It is a PPT algorithm that takes as input the security parameter* 1^λ *(encoded in unary) and a circuit* $C : \{0,1\}^n \rightarrow \{0,1\}^{m\lambda}$ *as input and outputs a garbled circuit* $\widetilde{C}$ *and state information* state.
- $\widetilde{x} \leftarrow \mathsf{SAdpGarbleInp}(\mathsf{state}, x)$: *It is a PPT algorithm that takes as input the state information* state *and an input* $x \in \{0,1\}^n$ *and outputs the garbled input* $\widetilde{x}$.
- $y = \mathsf{SAdpEvalCkt}(\widetilde{C}, \widetilde{x})$: *Given a garbled circuit* $\widetilde{C}$ *and a garbled input* $\widetilde{x}$*, it outputs a value* $y \in \{0,1\}^{m\lambda}$.

Correctness. *For every* $\lambda \in \mathbb{N}$, $C : \{0,1\}^n \rightarrow \{0,1\}^m$ *and* $x \in \{0,1\}^n$ *it holds that:*

$$\Pr\left[(\widetilde{C}, \mathsf{state}) \leftarrow \mathsf{SAdpGarbleCkt}(1^\lambda, C); \widetilde{x} \leftarrow \mathsf{SAdpGarbleInp}(\mathsf{state}, x) : C(x) = \mathsf{SAdpEvalCkt}(\widetilde{C}, \widetilde{x})\right] = 1.$$

Security. *There exists a PPT simulator* Sim *such that for all non-uniform PPT adversary* $\mathcal{A}$*:*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{Adp}}_{\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{Adp}}_{\mathcal{A}}(1^\lambda, 1) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{Adp}}_{\mathcal{A}}(1^\lambda, b)$ *is defined as follows:*

1. $(C, j) \leftarrow \mathcal{A}(1^\lambda)$ *where* $C : \{0,1\}^n \rightarrow \{0,1\}^{m\lambda}$ *and* $j \in [m]$*. We assume that* C *is given as a sequence of* m *circuits* $C_1, C_2, \ldots, C_m$.
2. *The adversary obtains* $\widetilde{C}$ *where* $\widetilde{C}$ *is created as follows:*
   - *If* $b = 0$*:* $(\widetilde{C}, \mathsf{state}) \leftarrow \mathsf{SAdpGarbleCkt}(1^\lambda, C)$.
   - *If* $b = 1$*:* $(\widetilde{C}, \mathsf{state}) \leftarrow \mathsf{Sim}(1^\lambda, C_1, \ldots, C_{j-1}, 1^{|C_j|}, C_{j+1}, \ldots, C_m)$.
3. *The adversary* $\mathcal{A}$ *specifies the input* x *and gets* $\widetilde{x}$ *created as follows:*
   - *If* $b = 0$: $\widetilde{x} \leftarrow \mathsf{SAdpGarbleInp}(\mathsf{state}, x)$.
   - *If* $b = 1$: $\widetilde{x} \leftarrow \mathsf{Sim}(\mathsf{state}, x, C_j(x))$.
4. *Finally, the adversary outputs a bit* b'*, which is the output of the experiment.*

Efficiency. *We require that the running time of* $\mathsf{SAdpGarbleInp}$ *to be* $\max_i |C_i| \cdot \mathsf{poly}(|x|, \lambda)$.

We give a construction of somewhere adaptive garbled circuits assuming the existence of one-way functions.

Lemma 3.2 *Assuming the existence of one-way functions, there exists a construction of somewhere adaptive garbled circuits.*

We give the proof of Lemma 3.2 in Appendix B.

3.3 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]). We provide a brief overview of the framework in Appendix A and refer the reader to [Can00b] for details.

3.4 Prior MPC Results

We will use the two-round secure multiparty computation protocol from the work of [GS18] computing special functionalities that have small circuit size in our constructions. We could also use the protocol from [BL18] but their protocol against malicious adversaries additionally relies on non-interactive zero-knowledge proofs. Below we restate the result from [GS18]. The ideal functionality $\mathcal{F}_f$ for the MPC is defined in Figure 2.

Theorem 3.3 ([GS18]) *For any polynomial-time function f computed by n parties, there exists a two-round UC-secure semi-honest/malicious multiparty computation protocol Π_f that realizes the ideal functionality $\mathcal{F}_f$, assuming the existence of semi-honest/malicious, two-round oblivious transfer. The number of total public key operations is bounded by $\mathsf{poly}(\lambda, |f|)$, where $|f|$ is the size of the Boolean circuit that computes f.*

$\mathcal{F}_f$ parameterized by a function f, running with n parties $P_1, P_2, \ldots, P_n$ (of which some may be corrupted) and an adversary $\mathcal{S}$, proceeds as follows:

- Every party P_i sends (sid, i, x_i) to the functionality.
- Upon receiving the inputs from all the parties, compute $y := f(x_1, \ldots, x_n)$, and output (sid, y) to every party and $\mathcal{S}$.

Figure 2: Ideal Functionality $\mathcal{F}_f$

4 Semi-Honest Protocol

In this section, we give a construction of two-round multiparty computation protocol with security against semi-honest adversaries that performs $\mathsf{poly}(n, \lambda)$ public key operations which is independent of the circuit size being computed. We start with the definition of conforming protocols which was a notion introduced in [GS18] in subsection 4.1 and then give our construction in subsection 4.2.

4.1 Conforming Protocols

This subsection is taken verbatim from [GS18]. Consider an n party deterministic[4] 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

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

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

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

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

  where pre is a randomized algorithm. The algorithm pre takes as input the index i of the party, its input x_i and outputs $z_i \in \{0,1\}^{\ell/n}$ and $v_i \in \{0,1\}^\ell$ (where ℓ is a parameter of the protocol). Finally, P_i retains v_i as the secret information and broadcasts z_i to every other party. We require that $v_i[k] = 0$ for all $k \in [\ell] \setminus \{(i-1)\ell/n + 1, \ldots, i\ell/n\}$.

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

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

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

  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 which 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 $\mathsf{post}(i, \mathsf{st}_i)$.

The following lemma was shown in [GS18]

Lemma 4.1 ([GS18]) *Any MPC protocol* Π *can be written as a conforming protocol* Φ *while inheriting the correctness and the security of the original protocol.*

4.2 Construction

In this subsection, we describe our construction of two-round, n-party computation protocol computing a function f. Our construction uses the following primitives.

1. An n-party semi-honest secure conforming protocol Φ computing the function f.

2. $(\mathsf{Garble}, \mathsf{Eval})$ be a garbling scheme for circuits.

3. A pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{4T}$.

4. A UC-secure two-round MPC protocol computing the function g described in Figure 3.

Parties: $P_1, P_2, \ldots, P_n$.
Inputs:

- P_1 (also called as the receiver) inputs $s \in \{0,1\}^\lambda$ and $\overline{\mathsf{rlab}}_2, \ldots, \overline{\mathsf{rlab}}_n$ where each $\overline{\mathsf{rlab}}_i$ is a collection of labels $\{\mathsf{rlab}_{j,0}^{i \to 1}, \mathsf{rlab}_{j,1}^{i \to 1}\}_{j \in [\lambda^2]}$ with each label of length λ.
- For each $i \in [2,n]$, P_i (also called as the sender) inputs $\overline{\mathsf{slab}}_i$, where $\overline{\mathsf{slab}}_i$ is a collection of labels $\{\mathsf{slab}_{j,0}^{i \to 1}, \mathsf{slab}_{j,1}^{i \to 1}\}_{j \in [\lambda]}$ with each label having length λ.

Output: $\{\mathsf{Projection}(\mathsf{Projection}(s, \overline{\mathsf{slab}}_i), \overline{\mathsf{rlab}}_i)\}_{i \in [2,n]}$.

Figure 3: The function g computed by the internal MPC where P_1 acts as the receiver

Notations. For a bit string c, we use $c[i]$ to denote the i-th bit of it. For each $t \in [T]$ and $\alpha, \beta \in \{0,1\}$, we use (t, α, β) to succinctly denote the integer $4t + 2\alpha + \beta - 3$. In particular, we use $c[(t,\alpha,\beta)]$ to denote $c[4t + 2\alpha + \beta - 3]$ for any $c \in \{0,1\}^{4T}$. We use $\overline{\mathsf{lab}}$ to denote the set of both labels per input wire of a garbled circuit, and $\widetilde{\mathsf{lab}}$ denotes the set of one label per input wire. Recall the definition of $\mathsf{Projection}$ from Section 3.

We give an overview of the construction below and describe the formal construction later.

Overview. As explained in Section 2, our construction combines a special purpose OT extension protocol (which is delegatable, fine-grained secure and satisfies key availability) along with the two-round MPC protocols of [BL18, GS18] to obtain a protocol that minimizes the number of public key operations. Recall that the protocols of [BL18, GS18] used the concept of "talking garbled circuits" to squish the round complexity of a conforming protocol to two rounds. At a high level, in the first round, every pair of parties sets up a channel to enable their garbled circuits to interact, and then in the second round, they send "talking garbled circuits" that emulate the interactions in the conforming protocol. The interaction between the "talking garbled circuits" is done via oblivious transfer. In our new construction, we use a special purpose OT extension protocol that allows the parties to set-up the channel for interaction while minimizing the number of public key operations.

A major modification from the description given in Section 2 is in modeling the special oblivious transfer as a protocol between a single receiver and $n-1$ senders. We do this to ensure that the receiver uses the same choice bits in interactions with every sender. Even though this is not an issue in the semi-honest case, it causes issues in the malicious setting if the corrupted receiver uses different choice bits in two different interactions. For uniformity of treatment, we adopt an approach where the special oblivious transfer is a protocol between a single receiver and $n-1$ senders.

Description of the Protocol. We give a formal description of our protocol below in the $\mathcal{F}_g$-hybrid model.

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 $t \in [T]$ and for each $\alpha, \beta \in \{0,1\}$

$$c_i[(t,\alpha,\beta)] := v_i[h] \oplus \mathsf{NAND}(v_i[f] \oplus \alpha, v_i[g] \oplus \beta)$$

where $\phi_t = (\star, f, g, h)$.

3. Sample $s_i \leftarrow \{0,1\}^\lambda$ and compute $e_i := \mathsf{PRG}(s_i) \oplus c_i$.
4. For each $j \in [n] \setminus \{i\}$, sample

$$\begin{aligned} \mathsf{rlab}_{k,b}^{j\to i} &\leftarrow \{0,1\}^\lambda \text{ for all } k \in [\lambda^2], b \in \{0,1\} \\ \mathsf{slab}_{k,b}^{i\to j} &\leftarrow \{0,1\}^\lambda \text{ for all } k \in [\lambda], b \in \{0,1\} \\ \mathsf{m}_{k,b}^{i\to j} &\leftarrow \{0,1\}^\lambda \text{ for all } k \in [4T], b \in \{0,1\} \end{aligned}$$

5. For each $j \in [n] \setminus \{i\}$, compute

$$\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j} \leftarrow \mathsf{Garble}\left(\mathsf{C}_{\mathsf{B}}\left[\left\{\mathsf{m}_{k,0}^{i\to j}, \mathsf{m}_{k,1}^{i\to j}\right\}_{k\in[4T],b\in\{0,1\}}\right], \left\{\mathsf{slab}_{k,b}^{i\to j}\right\}_{k\in[\lambda],b\in\{0,1\}}\right)$$

where C_{B} is described in Figure 4.

6. Send $(\mathsf{ssid} = i, s_i, \{\mathsf{rlab}_{k,b}^{j\to i}\}_{j\in[n]\setminus\{i\}})$ to $\mathcal{F}_g$ acting as the receiver.
7. For each $j \in [n] \setminus \{i\}$, send $(\mathsf{ssid} = j, \{\mathsf{slab}_{k,b}^{i\to j}\})$ to $\mathcal{F}_g$ acting as the sender.
8. Send $\left(z_i, \{\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}\}_{j\in[n]\setminus\{i\}}, e_i\right)$ to every other party.

$$\mathsf{C}_{\mathsf{B}}\left[\{\mathsf{m}_{k,0}, \mathsf{m}_{k,1}\}_{k\in[4T]}\right]$$

Input: $s \in \{0,1\}^\lambda$.

1. $d := \mathsf{PRG}(s)$ where $d \in \{0,1\}^{4T}$.
2. Output $\left\{\mathsf{m}_{k,d[k]}\right\}_{k\in[4T]}$.

Figure 4: Circuit C_{B}

Round-2: Each party P_i does the following:

1. Set $\mathsf{st}_i := (z_1\| \ldots \|z_n) \oplus v_i$.
2. Set $N = \ell + 4T\lambda(n-1)$.
3. Set $\overline{\mathsf{lab}}^{i,T+1} := \{\mathsf{lab}_{k,0}^{i,T+1}, \mathsf{lab}_{k,1}^{i,T+1}\}_{k\in[N]}$ where $\mathsf{lab}_{k,b}^{i,T+1} := 0^\lambda$ for each $k \in [N], b \in \{0,1\}$.
4. **for** each t from T down to 1 **do**:
   (a) Parse ϕ_t as (i^*, f, g, h).
   (b) If $i = i^*$ then compute (where P is described in Figure 6)

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

   (c) If $i \neq i^*$ then for every $\alpha, \beta \in \{0,1\}$, set $\mathsf{m}'_{\alpha,\beta,0} = \mathsf{m}^{i\to i^*}_{(t,\alpha,\beta),e_{i^*}[(t,\alpha,\beta)]}$ and $\mathsf{m}'_{\alpha,\beta,1} = \mathsf{m}^{i\to i^*}_{(t,\alpha,\beta),1\oplus e_{i^*}[(t,\alpha,\beta)]}$.
   Compute $ct^i_{\alpha,\beta} := (\mathsf{m}'_{\alpha,\beta,0} \oplus \mathsf{lab}_{h,0}^{i,t+1}, \mathsf{m}'_{\alpha,\beta,1} \oplus \mathsf{lab}_{h,1}^{i,t+1})$ and compute

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

5. Compute

$$\widetilde{\mathsf{C}}^{i}_{\mathsf{wrap}} \leftarrow \mathsf{Garble}\left(\mathsf{C}_{\mathsf{wrap}}\left[\{\widetilde{\mathsf{C}}^{j\to i}_{\mathsf{B}}\}_{j\in[n]\setminus\{i\}}, \mathsf{st}_i, \overline{\mathsf{lab}}^{i,1}\right], \{\mathsf{rlab}^{j\to i}_{k,b}\}_{j\in[n]\setminus\{i\}, k\in[\lambda^2], b\in\{0,1\}}\right)$$

   where $\mathsf{C}_{\mathsf{wrap}}$ is described in Figure 5.

6. Send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \widetilde{\mathsf{C}}^{i}_{\mathsf{wrap}}\right)$ to every other party.

$$\mathsf{C}_{\mathsf{wrap}}\left[\{\widetilde{\mathsf{C}}^{j\to i}_{\mathsf{B}}\}_{j\in[n]\setminus\{i\}}, \mathsf{st}_i, \overline{\mathsf{lab}}^{i,1}\right]$$

Input: $\{\widetilde{\mathsf{slab}}^{j\to i}\}_{j\in[n]\setminus\{i\}}$

1. For each $j \in [n]\setminus\{i\}$, compute $\left\{\mathsf{m}^{j\to i}_k\right\}_{k\in[4T]} \leftarrow \mathsf{Eval}\left(\widetilde{\mathsf{C}}^{j\to i}_{\mathsf{B}}, \widetilde{\mathsf{slab}}^{j\to i}\right)$.
2. Let $\mathsf{m} := \underset{j\in[n]\setminus\{i\}, k\in[4T]}{\|}(\mathsf{m}^{j\to i}_k)$.
3. Output $\mathsf{Projection}(\mathsf{st}_i\|\mathsf{m}, \overline{\mathsf{lab}}^{i,1})$.

Figure 5: Circuit $\mathsf{C}_{\mathsf{wrap}}$

Evaluation: Every party P_i does the following:

1. For each $j \in [n]$,
   (a) Obtain $(\mathsf{ssid} = j, \widetilde{\mathsf{rlab}}^{j})$ from $\mathcal{F}_g$ where party P_j acts as the receiver.
   (b) $\widetilde{\mathsf{lab}}^{j,1} \leftarrow \mathsf{Eval}(\widetilde{\mathsf{C}}^{j}_{\mathsf{wrap}}, \widetilde{\mathsf{rlab}}^{j})$
2. **for** each t from 1 to T **do**:
   (a) Parse ϕ_t as (i^*, f, g, h).
   (b) Compute $((\alpha,\beta,\gamma), \{\omega^j\}_{j\in[n]\setminus\{i^*\}}, \widetilde{\mathsf{lab}}^{i^*,t+1}) := \mathsf{Eval}(\widetilde{\mathsf{P}}^{i^*,t}, \widetilde{\mathsf{lab}}^{i^*,t})$.
   (c) Set $\mathsf{st}_i[h] := \gamma \oplus v_i[h]$.
   (d) **for** each $j \neq i^*$ **do**:
      i. Compute $(ct = (\delta_0,\delta_1), \{\mathsf{lab}^{j,t+1}_k\}_{k\in[N]\setminus\{h\}}) := \mathsf{Eval}(\widetilde{\mathsf{P}}^{j,t}, \widetilde{\mathsf{lab}}^{j,t})$.
      ii. Recover $\mathsf{lab}^{j,t+1}_h := \delta_\gamma \oplus \omega^j$.
      iii. Set $\widetilde{\mathsf{lab}}^{j,t+1} := \{\mathsf{lab}^{j,t+1}_k\}_{k\in[N]}$.
3. Compute the output as $\mathsf{post}(i, \mathsf{st}_i)$.

Correctness. In order to prove correctness, it is sufficient to show that the label $\mathsf{lab}^{j,t+1}_h$ 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 structure of v_{i^*} we have for every $j \neq i^*$, $\mathsf{st}_j[f] = \mathsf{st}_{i^*}[f] \oplus v_{i^*}[f]$.

First, ω^j is computed in Step 2b. Let $k := (t,\alpha,\beta)$, and we have $\omega^j = \mathsf{m}^{j\to i^*}_k = \mathsf{m}^{j\to i^*}_{k,\mathsf{PRG}(s_{i^*})[k]}$.

```
$\mathsf{P}\left[i, \phi_t, v_i, \{ct_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}, \overline{\mathsf{lab}}\right]$

Input. $Z = \Big(\mathsf{st}_i, \{\mathsf{m}_k^{j\to i}\}_{j\in[n]\setminus\{i\},k\in[4T]}\Big)$.
Hardcoded. The index $i$ of the party, the action $\phi_t = (i^*, f, g, h)$, the secret value $v_i$, the
strings $\{ct_{\alpha,\beta}\}_{\alpha,\beta\in\{0,1\}}$, and a set of labels $\overline{\mathsf{lab}} = \{\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}\}_{k\in[N]}$.

  1. if $i = i^*$ then:
     (a) Compute $\mathsf{st}_i[h] := \mathsf{NAND}(\mathsf{st}_i[f], \mathsf{st}_i[g])$, and update $Z[h]$ accordingly.
     (b) $\alpha := \mathsf{st}_i[f] \oplus v_i[f]$, $\beta := \mathsf{st}_i[g] \oplus v_i[g]$ and $\gamma := \mathsf{st}_i[h] \oplus v_i[h]$.
     (c) Output $\Big((\alpha,\beta,\gamma), \{\mathsf{m}^{j\to i}_{(t,\alpha,\beta)}\}_{j\in[n]\setminus\{i\}}, \mathsf{Projection}(Z, \overline{\mathsf{lab}})\Big)$.
  2. else:
     (a) Output $(ct_{\mathsf{st}_i[f],\mathsf{st}_i[g]}, \{\mathsf{lab}_{k,Z[k]}\}_{k\in[N]\setminus\{h\}})$.
```

Figure 6: The program P

Second, $ct = (\delta_0, \delta_1)$ is computed in Step 2(d)i. Note that $\alpha = \mathsf{st}_{i^*}[f] \oplus v_{i^*}[f] = \mathsf{st}_j[f]$, $\beta = \mathsf{st}_{i^*}[g] \oplus v_{i^*}[g] = \mathsf{st}_j[g]$. From the functionality of $\mathsf{P}^{j,t}$ we know that $ct = ct_{\mathsf{st}_j[f],\mathsf{st}_j[g]} = ct^j_{\alpha,\beta} = (\mathsf{m}'_{\alpha,\beta,0} \oplus \mathsf{lab}^{j,t+1}_{h,0}, \mathsf{m}'_{\alpha,\beta,1} \oplus \mathsf{lab}^{j,t+1}_{h,1}) = (\mathsf{m}^{j\to i^*}_{k,e_{i^*}[k]} \oplus \mathsf{lab}^{j,t+1}_{h,0}, \mathsf{m}^{j\to i^*}_{k,e_{i^*}[k]\oplus 1} \oplus \mathsf{lab}^{j,t+1}_{h,1})$.

Therefore, $\delta_\gamma \oplus \omega^j = \mathsf{m}^{j\to i^*}_{k,e_{i^*}[k]\oplus\gamma} \oplus \mathsf{lab}^{j,t+1}_{h,\gamma} \oplus \mathsf{m}^{j\to i^*}_{k,\mathsf{PRG}(s_{i^*})[k]}$. Recall that $c_{i^*}[k] = \mathsf{NAND}(\mathsf{st}_{i^*}[f], \mathsf{st}_{i^*}[g]) \oplus v_{i^*}[h] = \gamma$, thus $e_{i^*}[k] \oplus \gamma = e_{i^*}[k] \oplus c_{i^*}[k] = \mathsf{PRG}(s_{i^*})[k]$. Hence $\delta_\gamma \oplus \omega^j = \mathsf{lab}^{j,t+1}_{h,\gamma}$. This concludes the proof.

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 Z.

Efficiency. Let the number of public key operations in Φ be npk_Φ and in one execution of $\mathcal{F}_g$ be npk_g. We choose the conforming protocol that performs OT extension between every pair of parties so that npk_Φ is bounded by $\mathcal{O}(n^2\lambda)$. The total number of public key operations in our two-round construction is $\mathcal{O}(\mathsf{npk}_\Phi + n \cdot \mathsf{npk}_g)$. It follows from Theorems 3.3 that this number is bounded by $\mathsf{poly}(n, \lambda)$.

4.3 Simulator

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

Description of the Simulator. We give the description of the ideal world adversary S that simulates the view of the real world adversary $\mathcal{A}$. S will internally use the semi-honest simulator

Sim_Φ for Φ and the simulator $\mathsf{Sim}_{\mathsf{ckt}}$ for garbling scheme for circuits. Recall that $\mathcal{A}$ is static and hence the set of honest parties H is known before the execution of the protocol.

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

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

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

- **Round-1 messages from S to $\mathcal{A}$:** Next S generates $\widetilde{\mathsf{C}}_\mathsf{B}$ and e on behalf of honest parties as follows. For each $i \in H$:

  1. Sample $e_i \leftarrow \{0,1\}^{4T}$.
  2. For each $j \in [n] \setminus \{i\}$, sample $\mathsf{m}_k^{i \to j} \leftarrow \{0,1\}^\lambda$ for all $k \in [4T]$. Sample the $\mathsf{slab}_k^{i \to j} \leftarrow \{0,1\}^\lambda$ for each $k \in [\lambda]$ and compute
  $$\widetilde{\mathsf{C}}_\mathsf{B}^{i \to j} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \left\{\mathsf{m}_k^{i \to j}\right\}_{k \in [4T]}, \{\mathsf{slab}_k^{i \to j}\}_{k \in [\lambda]}\right).$$
  3. Send $\left(z_i, \{\widetilde{\mathsf{C}}_\mathsf{B}^{i \to j}\}_{j \in [n] \setminus \{i\}}, e_i\right)$ to the adversary $\mathcal{A}$ on behalf of the honest party P_i.

- **Round-1 messages from $\mathcal{A}$ to S:** Corresponding to every $i \in [n] \setminus H$, S receives from the adversary $\mathcal{A}$ the values $\left(z_i, \{\widetilde{\mathsf{C}}_\mathsf{B}^{i \to j}\}_{j \in [n] \setminus \{i\}}, e_i\right)$ on behalf of the corrupted party P_i. S recovers the value $\mathsf{m}_{k,b}^{i \to j}$ for all $j \in [n] \setminus \{i\}, k \in [4T], b \in \{0,1\}$ from random tape of party P_i.

- **Messages from $\mathcal{A}$ to $\mathcal{F}_g$:** For every $i \in [n] \setminus H$, S receives $(\mathsf{ssid} = i, s_i, \{\overline{\mathsf{rlab}}^{j \to i}\}_{j \in [n] \setminus \{i\}})$ and $\left\{(\mathsf{ssid} = j, \overline{\mathsf{slab}}^{i \to j})\right\}_{j \in [n] \setminus \{i\}}$ that sends to $\mathcal{F}_g$.

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

  1. Set $\widetilde{\mathsf{lab}}^{i,T+1} := \{\mathsf{lab}_k^{i,T+1}\}_{k \in [N]}$ where $\mathsf{lab}_k^{i,T+1} := 0^\lambda$ for each $k \in [N]$.

[5] 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.

2. **for** each t from T down to 1 **do**:
   (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 set $\mathsf{m}^{j \to i}_{(t,\alpha^*,\beta^*)} := \mathsf{m}^{j \to i}_{(t,\alpha^*,\beta^*),\gamma^* \oplus e_i[(t,\alpha^*,\beta^*)]}$ for each $j \in [n] \setminus H$, sample $\mathsf{lab}^{i,t}_k \leftarrow \{0,1\}^\lambda$ for each $k \in [N]$ and compute

   $$\widetilde{\mathsf{P}}^{i,t} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \left((\alpha^*, \beta^*, \gamma^*), \{\mathsf{m}^{j \to i}_{(t,\alpha^*,\beta^*)}\}_{j \in [n]\setminus\{i\}}, \{\mathsf{lab}^{i,t+1}_k\}_{k \in [N]}\right), \{\mathsf{lab}^{i,t}_k\}_{k \in [N]}\right).$$

   (d) If $i \neq i^*$, then set $\delta_{\gamma^*} := \mathsf{lab}^{i,t+1}_h \oplus \mathsf{m}^{i \to i^*}_{(t,\alpha^*,\beta^*)}$, sample $\delta_{1-\gamma^*} \leftarrow \{0,1\}^\lambda$, set $ct := (\delta_0, \delta_1)$, sample $\mathsf{lab}^{i,t}_k \leftarrow \{0,1\}^\lambda$ for each $k \in [N]$ and compute

   $$\widetilde{\mathsf{P}}^{i,t} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \left(ct, \{\mathsf{lab}^{i,t+1}_k\}_{k \in [N]\setminus\{h\}}\right), \{\mathsf{lab}^{i,t}_k\}_{k \in [N]}\right).$$

3. Sample $\mathsf{rlab}^i_k \leftarrow \{0,1\}^\lambda$ for each $k \in [(n-1)\lambda^2]$ (note that $(n-1)\lambda^2$ is the length of the input to $\mathsf{C}_{\mathsf{wrap}}$) and compute

   $$\widetilde{\mathsf{C}}^i_{\mathsf{wrap}} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \{\mathsf{lab}^{i,1}_k\}_{k \in [N]}, \{\mathsf{rlab}^i_k\}_{k \in [(n-1)\lambda^2]}\right).$$

4. Send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t \in [T]}, \widetilde{\mathsf{C}}^i_{\mathsf{wrap}}\right)$ to every other party.

- **Messages from $\mathcal{F}_g$ to $\mathcal{A}$:** For every $i \in H$, S sends $(\mathsf{ssid} = i, \{\mathsf{rlab}^i_k\}_{k \in [(n-1)\lambda^2]})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_g$. For every $i \in [n] \setminus H$, let $\widetilde{\mathsf{slab}}^{j \to i} := \mathsf{Projection}(s_i, \overline{\mathsf{slab}}^{j \to i})$ for each $j \in [n] \setminus H$; for each $j \in H$, define $\widetilde{\mathsf{slab}}^{j \to i} := \{\mathsf{slab}^{j \to i}_k\}_{k \in [\lambda]}$; S sends $(\mathsf{ssid} = i, \{\mathsf{Projection}(\widetilde{\mathsf{slab}}^{j \to i}, \overline{\mathsf{rlab}}^{j \to i})\}_{j \in [n]\setminus\{i\}})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_g$.

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

4.4 Proof of Indistinguishability

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

- $\mathcal{H}_0$: This hybrid is the same as the real world execution.

- $\mathcal{H}_1$: In this hybrid, the functionality $\mathcal{F}_g$ is emulated by the simulator S faithfully. This hybrid is identical to $\mathcal{H}_0$.

- $\mathcal{H}_2$: In this hybrid, we change how the $\mathcal{F}_g$ functionality is emulated by the simulator in every call where $i \in H$ is acting as the receiver. In particular, for any $\mathsf{ssid} = i \in H$, party P_i does not send $\left(s_i, \{\mathsf{rlab}^{j \to i}_{k,b}\}_{j \in [n]\setminus\{i\}}\right)$ to the ideal functionality. Instead, the simulator uses $\left(s_i, \{\mathsf{rlab}^{j \to i}_{k,b}\}_{j \in [n]\setminus\{i\}}\right)$ to directly compute the output. This hybrid is distributed identically to $\mathcal{H}_1$.

- $\mathcal{H}_3$: In this hybrid, we change how the $\mathcal{F}_g$ functionality is emulated by the simulator in every call where $i \in H$ is acting as the sender. In particular, for any $\mathsf{ssid} = j \in [n]$, when $i \in H$ is acting as the sender in this execution, party P_i does not send $\{\mathsf{slab}_{k,b}^{i\to j}\}$ to the ideal functionality. Instead, the simulator uses $\{\mathsf{slab}_{k,b}^{i\to j}\}$ to directly compute the output. This hybrid is distributed identically to $\mathcal{H}_2$.

- $\mathcal{H}_4$: In this hybrid we change $\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}$ sent from S to $\mathcal{A}$ on behalf of the honest parties $i \in H$ and the messages from $\mathcal{F}_g$ to $\mathcal{A}$ as follows.

  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. In hybrid $\mathcal{H}_4$ we make the following changes with respect to hybrid $\mathcal{H}_3$:

  - In Round 1, for each $i \in H$, S generates $\{\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}\}$ on behalf of party P_i as follows: For each $j \in [n] \setminus \{i\}, t \in [T], \alpha, \beta \in \{0,1\}$, let $k = (t, \alpha, \beta)$, set $\mathsf{m}_k^{i\to j} := \mathsf{m}_{k,\mathsf{PRG}(s_j)[k]}^{i\to j}$ where $\{\mathsf{m}_{k,0}^{i\to j}, \mathsf{m}_{k,1}^{i\to j}\}$ are the honestly generated masking strings. Sample the garbled circuit labels $\widetilde{\mathsf{slab}}^{i\to j}$ randomly and compute

    $$\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \left\{\mathsf{m}_k^{i\to j}\right\}_{k\in[4T]}, \widetilde{\mathsf{slab}}^{i\to j}\right).$$

  - **Messages from $\mathcal{F}_g$ to $\mathcal{A}$:** For each $i \in [n], j \in [n]\backslash H$: let $\widetilde{\mathsf{slab}}^{j\to i} := \mathsf{Projection}(s_i, \overline{\mathsf{slab}}^{j\to i})$. For each $i \in [n]$, S sends $(\mathsf{ssid} = i, \{\mathsf{Projection}(\widetilde{\mathsf{slab}}^{j\to i}, \overline{\mathsf{rlab}}^{j\to i})\}_{j\in[n]\backslash\{i\}})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_g$.

  The indistinguishability between $\mathcal{H}_3$ and $\mathcal{H}_4$ follows from $|H|$ invocations of the security of garbled circuits. In particular, for any $s, \mathsf{m}_{k,0}, \mathsf{m}_{k,1} \in \{0,1\}^\lambda$ for $k \in [4T]$, the following distributions are computationally indistinguishable:

  $$\begin{aligned} &\left\{\mathsf{Garble}\left(1^\lambda, \mathsf{C}_{\mathsf{B}}\left[\{\mathsf{m}_{k,0}, \mathsf{m}_{k,1}\}_{k\in[4T]}, \overline{\mathsf{slab}}\right]\right), \mathsf{Projection}(s, \overline{\mathsf{slab}})\right\} \\ \overset{c}{\approx} &\left\{\mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \mathsf{C}_{\mathsf{B}}(s), \widetilde{\mathsf{slab}}\right), \widetilde{\mathsf{slab}}\right\} \\ = &\left\{\mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \{\mathsf{m}_k\}_{k\in[4T]}, \widetilde{\mathsf{slab}}\right), \widetilde{\mathsf{slab}}\right\} \end{aligned}$$

  where $\overline{\mathsf{slab}}$ are randomly generated input keys for the garbled circuit (including both labels per input wire), and $\widetilde{\mathsf{slab}}$ are randomly generated labels for the garbled circuit (consisting of one label per input wire).

- $\mathcal{H}_5$: In this hybrid we change $\widetilde{\mathsf{C}}_{\mathsf{wrap}}^i$ sent from S to $\mathcal{A}$ on behalf of the honest parties $i \in H$ and the messages from $\mathcal{F}_g$ to $\mathcal{A}$ as follows.

  - In Round 2, for each $i \in H$, S generates $\widetilde{\mathsf{C}}_{\mathsf{wrap}}^i$ on behalf of party P_i as follows:

    1. For each $i \in [n] \setminus H, j \in [n] \setminus \{i\}$, compute $\mathsf{m}_k^{j\to i} := \mathsf{C}_{\mathsf{B}}^{j\to i}(s_i)$.
    2. Set $\mathsf{m} := \mathop{\|}\limits_{j\in[n]\backslash\{i\},k\in[4T]} (\mathsf{m}_k^{j\to i})$.

3. Sample the garbled circuit labels $\widetilde{\mathsf{rlab}}^i$ randomly and compute

$$\widetilde{\mathsf{C}}^i_{\mathsf{wrap}} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \mathsf{Projection}(\mathsf{st}_i \| \mathsf{m}, \overline{\mathsf{lab}}^{i,1}), \widetilde{\mathsf{rlab}}^i\right)$$

where $\overline{\mathsf{lab}}^{i,1}$ are the honestly generated input labels of $\widetilde{\mathsf{P}}^{i,1}$.

- **Messages from $\mathcal{F}_g$ to $\mathcal{A}$:** For every $i \in H$, S sends $(\mathsf{ssid} = i, \widetilde{\mathsf{rlab}}^i)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_g$. For every $i \in [n] \setminus H$, let $\widetilde{\mathsf{slab}}^{j\to i} := \mathsf{Projection}(s_i, \overline{\mathsf{slab}}^{j\to i})$ for each $j \in [n] \setminus H$; S sends $(\mathsf{ssid} = i, \{\mathsf{Projection}(\widetilde{\mathsf{slab}}^{j\to i}, \overline{\mathsf{rlab}}^{j\to i})\}_{j\in[n]\setminus\{i\}})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_g$.

The indistinguishability between $\mathcal{H}_4$ and $\mathcal{H}_5$ follows from $|H|$ invocations of the security of garbled circuits. In particular, for any $\{\widetilde{\mathsf{C}}^{j\to i}_{\mathsf{B}}\}_{j\in[n]\setminus\{i\}}, \mathsf{st}_i, \overline{\mathsf{lab}}^{i,1}, \{\widetilde{\mathsf{slab}}^{j\to i}\}_{j\in[n]\setminus\{i\}}$, we have

$$\begin{aligned}
&\left\{\mathsf{Garble}\left(1^\lambda, \mathsf{C}_{\mathsf{wrap}}\left[\{\widetilde{\mathsf{C}}^{j\to i}_{\mathsf{B}}\}_{j\in[n]\setminus\{i\}}, \mathsf{st}_i, \overline{\mathsf{lab}}^{i,1}\right], \{\overline{\mathsf{rlab}}^{j\to i}\}_{j\in[n]\setminus\{i\}}\right),\right.\\
&\quad\left.\{\mathsf{Projection}(\widetilde{\mathsf{slab}}^{j\to i}, \overline{\mathsf{rlab}}^{j\to i})\}_{j\in[n]\setminus\{i\}}\right\}\\
\overset{c}{\approx}&\left\{\mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \mathsf{C}_{\mathsf{wrap}}(\{\widetilde{\mathsf{slab}}^{j\to i}\}_{j\in[n]\setminus\{i\}}), \widetilde{\mathsf{rlab}}^i\right), \widetilde{\mathsf{rlab}}^i\right\}\\
=&\left\{\mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \mathsf{Projection}(\mathsf{st}_i \| \mathsf{m}, \overline{\mathsf{lab}}^{i,1}), \widetilde{\mathsf{rlab}}^i\right), \widetilde{\mathsf{rlab}}^i\right\}
\end{aligned}$$

where $\{\overline{\mathsf{rlab}}^{j\to i}\}_{j\in[n]\setminus\{i\}}$ are randomly generated input keys for the garbled circuit (including both labels per input wire), and $\widetilde{\mathsf{rlab}}^i$ are randomly generated labels for the garbled circuit (consisting of one label per input wire).

- $\mathcal{H}'_{5+t}$ (where $t \in \{1, \dots T\}$): The hybrids $\mathcal{H}'_{5+t}$ and $\mathcal{H}_{5+t}$ (which we will describe later) are the same as $\mathcal{H}_{4+t}$ except that we change 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 $\phi_t = (i^*, f, g, h)$. Let st^* be the local state at the end of the t-th round of execution, and set $\alpha^* := \mathsf{st}^*[f]$, $\beta^* := \mathsf{st}^*[g]$, and $\gamma^* := \mathsf{st}^*[h]$. Hybrid $\mathcal{H}'_{5+t}$ is the same as hybrid $\mathcal{H}_{4+t}$ except the changes below.

  - If $i^* \notin H$ then skip the changes. In the second round, S generates $\widetilde{\mathsf{P}}^{i^*,t}$ on behalf of P_{i^*}: Set $\mathsf{m}^{j\to i^*}_{(t,\alpha^*,\beta^*)} := \mathsf{m}^{j\to i^*}_{(t,\alpha^*,\beta^*),\gamma^*\oplus e_{i^*}[(t,\alpha^*,\beta^*)]}$ for each $j \in [n] \setminus H$. Compute m as in $\mathcal{H}_4$, sample the garbled circuit input labels $\widetilde{\mathsf{lab}}^{i^*,t}$ randomly, and compute

  $$\widetilde{\mathsf{P}}^{i^*,t} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \left((\alpha^*, \beta^*, \gamma^*), \{\mathsf{m}^{j\to i^*}_{(t,\alpha^*,\beta^*)}\}_{j\in[n]\setminus\{i^*\}}, \mathsf{Projection}(\mathsf{st}_{i^*} \| \mathsf{m}, \overline{\mathsf{lab}}^{i^*,t+1})\right), \widetilde{\mathsf{lab}}^{i^*,t}\right)$$

  where $\overline{\mathsf{lab}}^{i^*,t+1}$ are the honestly generated input labels of $\widetilde{\mathsf{P}}^{i^*,t+1}$.

  The indistinguishability between $\mathcal{H}'_{5+t}$ and $\mathcal{H}_{4+t}$ follows directly from the security of garbled circuits.

- $\mathcal{H}_{5+t}$ (where $t \in \{1, \dots T\}$): Hybrid $\mathcal{H}_{5+t}$ is the same as $\mathcal{H}'_{5+t}$ except the changes below.

- For each $i \in H$ and $i \neq i^*$, S generates $\widetilde{\mathsf{P}}^{i,t}$ on behalf of P_i: Set $\delta_{\gamma^*} := \mathsf{lab}_h^{i,t+1} \oplus \mathsf{m}^{i\to i^*}_{(t,\alpha^*,\beta^*),\gamma^*\oplus e_{i^*}[(t,\alpha^*,\beta^*)]}$, sample $\delta_{1-\gamma^*} \leftarrow \{0,1\}^\lambda$, set $ct := (\delta_0, \delta_1)$. Compute m as in $\mathcal{H}_4$ and $\widetilde{\mathsf{lab}}^{i,t+1} := \mathsf{Projection}(\mathsf{st}_i \| \mathsf{m}, \overline{\mathsf{lab}}^{i,t+1})$ where $\overline{\mathsf{lab}}^{i,t+1}$ are the honestly generated input labels of $\widetilde{\mathsf{P}}^{i,t+1}$. Sample the garbled circuit input labels $\widetilde{\mathsf{lab}}^{i,t}$ randomly and compute

$$\widetilde{\mathsf{P}}^{i,t} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \left(ct, \{\mathsf{lab}_k^{i,t+1}\}_{k\in[N]\setminus\{h\}}\right), \widetilde{\mathsf{lab}}^{i,t}\right).$$

Notice that here we sample $\delta_{1-\gamma^*} \leftarrow \{0,1\}^\lambda$ instead of $\mathsf{lab}_h^{i,t+1} \oplus \mathsf{m}^{i\to i^*}_{(t,\alpha^*,\beta^*),1\oplus\gamma^*\oplus e_{i^*}[(t,\alpha^*,\beta^*)]}$. Since $\mathsf{m}^{i\to i^*}_{(t,\alpha^*,\beta^*),1\oplus\gamma^*\oplus e_{i^*}[(t,\alpha^*,\beta^*)]}$ never appears anywhere else, $\mathsf{lab}_h^{i,t+1}$ is information theoretically hidden, hence changing $\delta_{1-\gamma^*}$ to be randomly sampled does not change the distribution. The rest of the indistinguishability proof between $\mathcal{H}_{5+t}$ and $\mathcal{H}'_{5+t}$ follows from the security of garbled circuits.

- $\mathcal{H}_{6+T}$: Hybrid $\mathcal{H}_{6+T}$ is the same as $\mathcal{H}_{5+T}$ except that in the first round, S samples $e_i \leftarrow \{0,1\}^{4T}$ for each $i \in H$ instead of $\mathsf{PRG}(s_i) \oplus c_i$.

  Notice that in hybrid $\mathcal{H}_{5+T}$, for each $i \in H$, s_i and c_i is only used to generate e_i. Hence the indistinguishability between $\mathcal{H}_{6+T}$ and $\mathcal{H}_{5+T}$ follows from $|H|$ invocations of the security of the pseudorandom generator PRG.

- $\mathcal{H}_{7+T}$: In this hybrid we just change how the transcript Z, $\{z_i\}_{i\in H}$, random coins of corrupted 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 $\mathcal{H}_{7+T}$ and $\mathcal{H}_{6+T}$ follows directly from the semi-honest security of the protocol Φ. Finally note that $\mathcal{H}_{6+T}$ is same as the ideal execution (i.e., the simulator described in the previous subsection).

5 Special Zero-Knowledge Protocol

In this section, we define and construct a special zero-knowledge protocol which will later be used in our construction against malicious adversaries. We give the formal definition below.

Definition 5.1 *A special zero-knowledge protocol is a two-round protocol that securely realizes the $\mathcal{F}_{\mathsf{ZK}}$ functionality given in Figure 7. Further, we require the number of pubic key operations performed in the protocol to be bounded by* $\mathsf{poly}(n, \lambda)$ *independent of the size of x and w.*

We give a proof of the following theorem.

Theorem 5.2 *Assuming the existence of two-round UC secure oblivious transfer, there exists a construction of special zero-knowledge protocol.*

5.1 Construction

We first describe the tools used in the construction.

$\mathcal{F}_{\mathsf{ZK}}$ parameterized by an NP relation R, running with n parties $P_1, P_2, \ldots, P_n$ (of which some may be corrupted) and an adversary $\mathcal{S}$, proceeds as follows:

- P_1 sends $(\mathsf{prover}, \mathsf{sid}, x, w)$ to the functionality. The functionality sends $(\mathsf{request}, x, R(x, w))$ to $\mathcal{S}$. If $\mathcal{S}$ has corrupted P_2, then $\mathcal{S}$ sends $(\mathsf{response}, \mu)$ to the ideal functionality, and the ideal functionality broadcasts $(R(x, w), x, \mu)$ to every other party and goes offline. Else, P_2 sends $(\mathsf{verifier}, \mathsf{sid}, \mu_0, \mu_1)$ to the functionality, where $\mu_b \in \{0, 1\}^\lambda$.
- Upon receiving the inputs from both P_1 and P_2, functionality checks if $R(x, w) = 1$. If yes, it sends $(1, x, \mu_1)$ to every party. Otherwise, it sends $(0, x, \mu_0)$ to all parties.

Figure 7: Special Zero-Knowledge Functionality $\mathcal{F}_{\mathsf{ZK}}$

1. **Special Non-interactive Statistically Binding Commitment.** We use a special non-interactive, statistically binding commitment scheme $(\mathsf{com}, \mathsf{decom})$ where the length of the randomness used to commit to arbitrary length messages is λ. We note that any standard commitment can be made to satisfy this property by using a pseudorandom generator to expand the random string to required length.

2. **Blum's Hamiltonicity Protocol.** We use the three-round, constant soundness zero-knowledge $(\mathsf{zk}_1, \mathsf{zk}_2, \mathsf{zk}_3)$ protocol of Blum. We note that in Blum's protocol $\mathsf{zk}_2 \in \{0, 1\}$ and we let $\mathsf{zk}_{3,b}$ be the response when $\mathsf{zk}_2 = b$. We also assume without loss of generality that zk_1 includes the instance.

3. **Two-Round Secure Computation Protocol.** We make use of the two-round secure computation protocol of [GS18] (that can be based on any two-round UC secure oblivious transfer) computing the ideal functionality $\mathcal{F}_f$ described in Figure 10.

4. **Length Doubling Pseudorandom Generator:** We use a pseudorandom generator $\mathsf{PRG} : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda}$.

Overview. We give an overview of our construction of the two-round special zero-knowledge protocol below, and present the formal construction in the $\mathcal{F}_f$ hybrid model in Figure 8.

In the first round, the prover computes Blum's proof $\mathsf{zk}_1, \mathsf{zk}_{3,0}, \mathsf{zk}_{3,1}$ for the proof w, and computes the commitment (c_0, c_1) of $(\mathsf{zk}_{3,0}, \mathsf{zk}_{3,1})$ using randomness (r_0, r_1) respectively. It then sends $(\mathsf{zk}_1, c_0, c_1)$ to the verifier. In the second round, the verifier samples a random bit $b \leftarrow \{0, 1\}$, and constructs a circuit C that has $(b, \mathsf{zk}_1, c_0, c_1, \mu_0, \mu_1)$ hardwired in it. The circuit C takes r as input, checks if r decommits c_b and if $(\mathsf{zk}_1, b, \mathsf{zk}_b)$ is a valid proof. If both checks pass, then it outputs μ_1; otherwise it outputs μ_0. The verifier garbles the circuit C and sends $\widetilde{C}$ to the prover.

Additionally, the prover and verifier in parallel run a two-round MPC protocol to compute the labels for evaluating the garbled circuit $\widetilde{C}$. Specifically, they jointly compute a function f which takes as input (r_0, r_1) from the prover and b along with the input labels for $\widetilde{C}$ from the verifier, and outputs the labels that correspond to r_b. Thus, at the end of the protocol, all parties can obtain the labels for evaluating $\widetilde{C}$ and compute the output of C.

Common Random String: Sample $\sigma \leftarrow \{0,1\}^{2\lambda}$ and set σ as the CRS.

Message from P_1: On input an instance x and a witness w, P_1 does the following:

1. If $R(x, w) = 0$, broadcast $(\mathsf{NotInL}, x, R(x, w))$ to every other party.
2. Else, **for** each $i \in [\lambda]$ **do**:
   (a) Prepare zk_1^i for the language $\mathcal{L}$ using the witness w where $\mathcal{L}$ is defined below.
   $$\mathcal{L} := \{(x, \sigma) : \exists\ (w, s) \text{ s.t. } R(x, w) = 1 \vee \mathsf{PRG}(s) = \sigma\}$$
   (b) Let $\mathsf{zk}_{3,b}^i$ be the third round message when $\mathsf{zk}_2^i = b$. Sample $r_b^i \leftarrow \{0,1\}^\lambda$ for each $b \in \{0,1\}$ and compute $c_b^i := \mathsf{com}(\mathsf{zk}_{3,b}^i; r_b^i)$.
   (c) Broadcast $\mathsf{zk}_1^i, c_0^i, c_1^i$ to every other party.

Message from P_2: On input the message from P_1 :

1. If P_1 has sent $(\mathsf{NotInL}, x, 0)$, broadcast μ_0 to every other party and every party outputs $(0, x, \mu_0)$. Else, do:
   (a) Sample $ch \leftarrow \{0,1\}^\lambda$.
   (b) Sample $\mathsf{lab}_{w,b}^i \leftarrow \{0,1\}^\lambda$ for each $i, w \in [\lambda]$ and $b \in \{0,1\}$.
   (c) Compute $\widetilde{C} \leftarrow \mathsf{Garble}(C[ch, \{\mathsf{zk}_1^i, c_0^i, c_1^i\}_{i\in[\lambda]}, \{\mu_b\}_{b\in\{0,1\}}], \{\mathsf{lab}_{w,b}^i\})$ where the C is described in Figure 9.
   (d) Broadcast $\widetilde{C}$ to every party.

Internal MPC: The parties in parallel call $\mathcal{F}_f$ to jointly compute the function f shown in Figure 10. More specifically, P_1 sends $\{r_0^i, r_1^i\}_{i\in[\lambda]}$ to $\mathcal{F}_f$; P_2 sends $ch, \{\mathsf{lab}_{w,b}^i\}_{i,w\in[\lambda],b\in\{0,1\}}$ to $\mathcal{F}_f$; and $P_3, P_4, \ldots, P_n$ send nothing. Every party then gets $\{\mathsf{lab}_w^i\}_{i,w\in[\lambda]}$ back from $\mathcal{F}_f$.

Evaluation: Every party does the following:

1. Compute $(b, x, \mu) \leftarrow \mathsf{Eval}\left(\widetilde{C}, \{\mathsf{lab}_w^i\}_{i,w\in[\lambda]}\right)$
2. Output (b, x, μ).

Figure 8: Special Zero-Knowledge Protocol Π_{ZK}

In order to achieve negligible soundness, the above protocol is repeated λ times in parallel, and C has all the λ sets of messages hardwired in it. However, it is well-known that parallel repetition does not preserve zero-knowledge but preserves witness indistinguishability [FS90]. To achieve zero-knowledge, we change the language $\mathcal{L}$ to include an additional "trapdoor" condition that $\mathsf{PRG}(s) = \sigma$, where σ is a part of the common reference string. In the proof of zero-knowledge, we will use the trapdoor witness s.

$C\left[ch, \{\mathsf{zk}_1^i, c_0^i, c_1^i\}_{i\in[\lambda]}, \{\mu_b\}_{b\in\{0,1\}}\right]$
Input: $r^1, r^2, \ldots, r^\lambda$.
Hardcoded parameters: $ch, \{\mathsf{zk}_1^i, c_0^i, c_1^i\}_{i\in[\lambda]}, \{\mu_b\}_{b\in\{0,1\}}$

1. Use the randomness r^i to obtain the message zk_3^i committed in $c_{ch[i]}^i$ for each $i \in [\lambda]$.
2. For each $i \in [\lambda]$, check if $(\mathsf{zk}_1^i, ch[i], \mathsf{zk}_3^i)$ is a valid proof for the membership in language $\mathcal{L}$.
3. If any of the checks fails, output $(0, x, \mu_0)$. Else, output $(1, x, \mu_1)$.

Figure 9: Circuit C

Parties: $P_1, P_2, \ldots, P_n$.
Inputs:

- P_1 inputs $\{r_0^i, r_1^i\}_{i\in[\lambda]}$, where $r_i^b \in \{0,1\}^\lambda$.
- P_2 inputs $ch, \{\mathsf{lab}_{w,b}^i\}_{i,w\in[\lambda],b\in\{0,1\}}$, where $\mathsf{lab}_{w,b}^i \in \{0,1\}^\lambda$.
- $P_3, P_4, \ldots, P_n$ input nothing.

Output: $\{\mathsf{lab}^i_{w,r^i_{ch[i]}[w]}\}_{i,w\in[\lambda]}$ (same for every party).

Figure 10: The Function f Computed by the Internal MPC

Correctness. To argue the correctness of the protocol, we only need to prove that in the evaluation step, μ is either μ_0 or μ_1 based on whether $R(x, w) = 0$ or $R(x, w) = 1$. We know that the output of $\mathcal{F}_f$ is $\left\{\mathsf{lab}_w^i\right\}_{i,w\in[\lambda]}$, where $\mathsf{lab}_w^i = \mathsf{lab}^i_{w,r^i_{ch[i]}[w]}$. Notice that $\mathsf{lab}_{w,b}^i$'s are the input keys of $\widetilde{C}$, hence lab_w^i is the label corresponding to the w-th bit of $r^i_{ch[i]}$. Using these input labels to evaluate $\widetilde{C}$ gives us $\mathsf{Eval}\left(\widetilde{C}, \left\{\mathsf{lab}_w^i\right\}_{i,w\in[\lambda]}\right) = C\left(\left\{r^i_{ch[i]}\right\}_{i\in[\lambda]}\right)$.

In the circuit evaluation of C, $r^i_{ch[i]}$ is used to obtain $\mathsf{zk}^i_{3,ch[i]}$ from $c^i_{ch[i]}$. It now follows from the completeness of $(\mathsf{zk}_1^i, ch[i], \mathsf{zk}^i_{3,ch[i]})$ that μ is either μ_0 or μ_1 based on $R(x, w) = 0$ or $R(x, w) = 1$.

Efficiency. The number of public key operations performed in the protocol is $\mathsf{poly}(n, \lambda)$ which follows from Theorem 3.3 when applied to function f.

5.2 Security

We consider three cases: either P_2 is corrupted, or P_1 is corrupted, or both P_1 and P_2 are not corrupted. In each of these cases, we describe ideal world simulators and show that no environment can distinguish between the real world and the ideal world in each of these cases.

5.2.1 Case-1: P_2 is corrupted

We first show the construction of an ideal world simulator in Figure 11 and then show that the real world distribution is computationally indistinguishable to the ideal world.

Common Random String: Sample $s \leftarrow \{0,1\}^\lambda$ and set $\sigma := \mathsf{PRG}(s)$ as the CRS.

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

- **Message from Sim:** Sim does the following:
  1. Sim obtains $(\mathsf{request}, x, R(x,w))$ from $\mathcal{F}_{\mathsf{ZK}}$. If $R(x,w) = 0$ send $(\mathsf{NotInL}, x, 0)$ to every party and obtain μ from P_2. Send $(\mathsf{response}, \mu)$ to $\mathcal{F}_{\mathsf{ZK}}$. Else, do the following.
  2. **for** each $i \in [\lambda]$ **do**:
     (a) Prepare zk_1^i for the language $\mathcal{L}$ using the witness s.
     (b) Let $\mathsf{zk}_{3,b}^i$ be the third round message when $\mathsf{zk}_2 = b$. Sample $r_b^i \leftarrow \{0,1\}^\lambda$ for each $b \in \{0,1\}$ and compute $c_b^i := \mathsf{com}(\mathsf{zk}_{3,b}^i; r_b^i)$.
     (c) Broadcast $\mathsf{zk}_1^i, c_0^i, c_1^i$ to every other party.
- **Internal MPC:** The parties in parallel call $\mathcal{F}_f$ to jointly compute the function f shown in Figure 10. More specifically, Sim sends $\{r_0^i, r_1^i\}_{i\in[\lambda]}$ to $\mathcal{F}_f$; $\mathcal{A}$ sends $ch, \{\mathsf{lab}_{w,b}^i\}_{i,w\in[\lambda],b\in\{0,1\}}$ to $\mathcal{F}_f$; and $P_3, P_4, \ldots, P_n$ send nothing. Every party then gets $\{\mathsf{lab}_w^i\}_{i,w\in[\lambda]}$ back from $\mathcal{F}_f$.
- **Message from $\mathcal{A}$:** Receive the message $\widetilde{C}$ from $\mathcal{A}$. Compute $(b, x, \mu) \leftarrow \mathsf{Eval}\left(\widetilde{C}, \{\mathsf{lab}_w^i\}_{i,w\in[\lambda]}\right)$. Send $(\mathsf{response}, \mu)$ to $\mathcal{F}_{\mathsf{ZK}}$.

Figure 11: Simulator Sim when P_2 is corrupted

We now show that the simulated distribution is computationally indistinguishable to the honest prover's distribution. We show this through a sequence of hybrids.

$\underline{\mathcal{H}_1}$: This hybrid is same as the honest prover distribution except that we generate the common random string σ as $\mathsf{PRG}(s)$ where s is chosen uniformly from $\{0,1\}^\lambda$. $\mathcal{H}_1$ is computationally indistinguishable from the honest prover's execution from the security of PRG.

If $R(x,w) = 0$, then $\mathcal{H}_1$ is identical to the ideal world distribution generated by Sim. In the rest of the proof, we consider the case where $R(x,w) = 1$.

We now define a sequence of hybrids, specifically, one hybrid for each $i \in [\lambda+1]$.

$\underline{\mathcal{H}_{2,i}}$: In this hybrid, for each $j \in [i-1]$, we use the witness s to generate $\mathsf{zk}_1^j, \mathsf{zk}_{3,0}^j, \mathsf{zk}_{3,1}^j$ and for $j \in [i, \lambda]$, we use the witness w. Note that $\mathcal{H}_{2,1}$ is distributed identically to $\mathcal{H}_1$ and $\mathcal{H}_{2,\lambda+1}$ is distributed identically to the simulated execution given in Figure 11. We now argue that $\mathcal{H}_{2,i-1}$ is indistinguishable to $\mathcal{H}_{2,i}$ for every $i \in [\lambda + 1]$.

Lemma 5.3 *Assuming the hiding property of the commitment scheme and witness indistinguishability property of Blum's protocol, $\mathcal{H}_{2,i-1} \overset{c}{\approx} \mathcal{H}_{2,i}$ for every $i \in [\lambda + 1]$.*

Proof We first define an intermediate hybrid $\mathcal{H}_{2,i-1,1}$.

$\underline{\mathcal{H}_{2,i-1,1}}$: In this hybrid, we generate the first round message from P_1 and the messages to $\mathcal{F}_f$ as follows:

```
1. Set $\mathsf{counter} = 0$.
2. while($\mathsf{counter} \leq \lambda$) :
   Message from $P_1$: On input the instance $x$ and witness $w$:
      (a) for each $j \in [\lambda] \setminus \{i-1\}$ do:
            i. Prepare $\mathsf{zk}_1^j, c_0^j, c_1^j$ as in $\mathcal{H}_{2,i-1}$ and broadcast it to every party.
      (b) for $j = i - 1$ do :
            i. Prepare $\mathsf{zk}_1^j, \mathsf{zk}_{3,0}^j, \mathsf{zk}_{3,1}^j$ as in $\mathcal{H}_{2,i-1}$.
            ii. Choose a random bit $b[j]$. Generate $c_{b[j]}^j$ as $\mathsf{com}(\mathsf{zk}_{3,b[j]}^j; r_{b[j]}^j)$ and $c_{1-b[j]}^j$ as $\mathsf{com}(0^{|\mathsf{zk}_{3,1-b[j]}^j|}; r_{1-b[j]}^j)$.
            iii. Broadcast $\mathsf{zk}_1^j, c_0^j, c_1^j$ to every other party.
   Internal MPC: (a) $\mathsf{Sim}$ resets $r_{1-b[i-1]}^{i-1} := 0^\lambda$.
      (b) The parties in parallel call $\mathcal{F}_f$ to jointly compute the function $f$ shown in Figure 10. More specifically, $\mathsf{Sim}$ sends $\{r_0^j, r_1^j\}_{j \in [\lambda]}$ to $\mathcal{F}_f$; $P_2$ sends $ch, \{\mathsf{lab}_{w,b}^i\}_{i,w \in [\lambda], b \in \{0,1\}}$ to $\mathcal{F}_f$; and $P_3, P_4, \ldots, P_n$ send nothing. If $ch[i-1] \neq b[i-1]$, send $\mathsf{abort}$ to $\mathcal{F}_f$, increment $\mathsf{counter}$ by 1 and go to the beginning of the while loop. Else exit the while loop.
3. if($\mathsf{counter} > \lambda$), output $\mathsf{special-abort}$.
```

We now argue that $\mathcal{H}_{2,i-1}$ is indistinguishable to $\mathcal{H}_{2,i-1,1}$ assuming the hiding property of the commitment scheme.

Claim 5.4 *Assuming the hiding property of the commitment scheme, $\mathcal{H}_{2,i-1} \overset{c}{\approx} \mathcal{H}_{2,i-1,1}$.*

Proof We show this via an intermediate hybrid H_1.

$\underline{\mathsf{H}_1}$: In this hybrid, we generate the first round message from P_1 and the messages to $\mathcal{F}_f$ as follows:

```
1. Set $\mathsf{counter} = 0$.
2. while($\mathsf{counter} \leq \lambda$) :
   Message from $P_1$: On input the instance $x$ and witness $w$:
```

(a) Choose a random bit $b[i-1]$.

(b) **for** each $j \in [\lambda]$ **do**:

- i. Prepare $\mathsf{zk}_1^j, c_0^j, c_1^j$ as in $\mathcal{H}_{2,i-1}$ and broadcast it to every party.

Internal MPC: (a) Sim resets $r^{i-1}_{1-b[i-1]} := 0^\lambda$.

(b) The parties in parallel call $\mathcal{F}_f$ to jointly compute the function f shown in Figure 10. More specifically, Sim sends $\{r_0^j, r_1^j\}_{j\in[\lambda]}$ to $\mathcal{F}_f$; P_2 sends $ch, \{\mathsf{lab}^i_{w,b}\}_{i,w\in[\lambda],b\in\{0,1\}}$ to $\mathcal{F}_f$; and $P_3, P_4, \ldots, P_n$ send nothing. If $ch[i-1] \neq b[i-1]$, send abort to $\mathcal{F}_f$, increment $\mathsf{counter}$ by 1 and go to the beginning of the **while** loop. Else exit the **while** loop.

3. **if**$(\mathsf{counter} > \lambda)$, output $\mathsf{special-abort}$.

The only difference between H_1 and $\mathcal{H}_{2,i-1}$ is that H_1 may output $\mathsf{special-abort}$. We show that the probability that H_1 outputs $\mathsf{special-abort}$ is negligible. Notice that in any execution of the **while** loop, the probability (over the choice of $b[i-1]$) that abort is sent to $\mathcal{F}_f$ is 1/2. Thus, the probability that $\mathsf{special-abort}$ is output by H_1 is $2^{-\lambda}$.

The only difference between H_1 and $\mathcal{H}_{2,i-1,1}$ is that in $\mathcal{H}_{2,i-1,1}$, $c^{i-1}_{1-b[i-1]}$ is generated by $\mathsf{com}(0^{|\mathsf{zk}^{i-1}_{3,1-b[i-1]}|}; r^{i-1}_{1-b[i-1]})$. It follows from the hiding property of com that H_1 is computationally indistinguishable to $\mathcal{H}_{2,i-1,1}$.

Thus, $\mathcal{H}_{2,i-1} \overset{s}{\approx} \mathsf{H}_1 \overset{c}{\approx} \mathcal{H}_{2,i-1,1}$. This completes the proof of the claim. ■

We now define another hybrid $\mathcal{H}_{2,i-1,2}$.

$\underline{\mathcal{H}_{2,i-1,2}}$:We generate the first round message from P_1 and the messages to $\mathcal{F}_f$ as follows.

1. Set $\mathsf{counter} = 0$.
2. **while**$(\mathsf{counter} \leq \lambda)$:

Message from P_1: On input the instance x and witness w:

(a) **for** each $j \in [\lambda] \setminus \{i-1\}$ **do**:

- i. Prepare $\mathsf{zk}_1^j, c_0^j, c_1^j$ as in $\mathcal{H}_{2,i-1}$ and broadcast it to every party.

(b) **for** $j = i-1$ **do** :

- i. Prepare $\mathsf{zk}_1^j, \mathsf{zk}_{3,0}^j, \mathsf{zk}_{3,1}^j$ using the witness s (instead of w).
- ii. Choose a random bit $b[j]$. Generate $c^j_{b[j]}$ as $\mathsf{com}(\mathsf{zk}^j_{3,b[j]}; r^j_{b[j]})$ and $c^j_{1-b[j]}$ as $\mathsf{com}(0^{|\mathsf{zk}^j_{3,1-b[j]}|}; r^j_{1-b[j]})$.
- iii. Broadcast $\mathsf{zk}_1^j, c_0^j, c_1^j$ to every other party.

Internal MPC: (a) Sim resets $r^{i-1}_{1-b[i-1]} := 0^\lambda$.

(b) The parties in parallel call $\mathcal{F}_f$ to jointly compute the function f shown in Figure 10. More specifically, Sim sends $\{r_0^j, r_1^j\}_{j\in[\lambda]}$ to $\mathcal{F}_f$; P_2 sends $ch, \{\mathsf{lab}^i_{w,b}\}_{i,w\in[\lambda],b\in\{0,1\}}$ to $\mathcal{F}_f$; and $P_3, P_4, \ldots, P_n$ send nothing. If $ch[i-1] \neq b[i-1]$, send abort to $\mathcal{F}_f$, increment $\mathsf{counter}$ by 1 and go to the beginning of the **while** loop. Else exit the **while** loop.

3. **if**$(\mathsf{counter} > \lambda)$, output $\mathsf{special-abort}$.

It follows from the witness indistinguishability property of Blum's protocol that $\mathcal{H}_{2,i-1,2}$ is computationally indistinguishable to $\mathcal{H}_{2,i-1,1}$.

We can show via an identical argument to Claim 5.4 that $\mathcal{H}_{2,i-1,2}$ is computationally indistinguishable to $\mathcal{H}_{2,i}$. This completes the proof of the lemma. ■

5.2.2 Case-2: P_1 is corrupted

We first describe the ideal world simulator in Figure 12.

Common Random String: Sample $\sigma \leftarrow \{0,1\}^{2\lambda}$ and set σ as the CRS.

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

- **Message from $\mathcal{A}$:** If $\mathcal{A}$ sends $(\mathsf{NotInL}, x, 0)$ then sample a random w and send $(\mathsf{prover}, \mathsf{sid}, x, w)$ to $\mathcal{F}_{\mathsf{ZK}}$. Else, receive the first round message $\{\mathsf{zk}_1^i, c_0^i, c_1^i\}_{i\in[\lambda]}$ from $\mathcal{A}$.
- **Internal MPC:** Recover the message $\{r_0^i, r_1^i\}_{i\in[\lambda]}$ that $\mathcal{A}$ sends to $\mathcal{F}_f$. For every $i \in [\lambda]$, use r_0^i, r_1^i to recover $\mathsf{zk}_{3,0}^i, \mathsf{zk}_{3,1}^i$ from c_0^i, c_1^i. Choose a random $ch \leftarrow \{0,1\}^\lambda$ and check if $(\mathsf{zk}_1^i, ch[i], \mathsf{zk}_{3,ch[i]}^i)$ is a valid proof. If any of the checks fail, sample a random w and send $(\mathsf{prover}, \mathsf{sid}, x, w)$ to $\mathcal{F}_{\mathsf{ZK}}$. Else, check if for any $i \in [\lambda]$ both $(\mathsf{zk}_1^i, 0, \mathsf{zk}_{3,0}^i)$ and $(\mathsf{zk}_1^i, 1, \mathsf{zk}_{3,1}^i)$ are valid proofs. If yes, recover w from these two transcripts and send $(\mathsf{prover}, \mathsf{sid}, x, w)$ to $\mathcal{F}_{\mathsf{ZK}}$. Else, output $\mathsf{special} - \mathsf{abort}$.
- **Message from Sim:** Sim does the following:
  1. Receive μ from $\mathcal{F}_{\mathsf{ZK}}$ and send μ to every party if $\mathcal{A}$ sends $(\mathsf{NotInL}, x, 0)$.
  2. Sample $\mathsf{lab}_w^i \leftarrow \{0,1\}^\lambda$ for each $i, w \in [\lambda]$.
  3. Compute $\widetilde{C} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, 1^{|C|}, \mu, \{\mathsf{lab}_w^i\}_{i,w\in\mathsf{inp}(C)}\right)$
  4. Broadcast $\widetilde{C}$ and send $\{\mathsf{lab}_w^i\}_{i,w\in[\lambda]}$ as the output from $\mathcal{F}_f$ to every party.

Figure 12: Simulator Sim when P_1 is corrupted

We now argue that the distribution generated by Sim is computationally indistinguishable to the honest P_2's distribution. We first note that since σ is chosen randomly from $\{0,1\}^{2\lambda}$, with overwhelming probability there exists no $s \in \{0,1\}^\lambda$ such that $\mathsf{PRG}(s) = \sigma$. In the rest of the proof, we condition on this event happening.

Claim 5.5 *Assuming the security of garbled circuits, we have that the real world distribution is computationally indistinguishable to the ideal world.*

Proof We first argue that the probability that the ideal simulator Sim outputs $\mathsf{special} - \mathsf{abort}$ is negligible. Note that Sim outputs $\mathsf{special} - \mathsf{abort}$ when for a random ch, $(\mathsf{zk}_1^i, ch[i], \mathsf{zk}_{3,ch[i]}^i)$ is a valid proof but for every $i \in [\lambda]$, $(\mathsf{zk}_1^i, 1 - ch[i], \mathsf{zk}_{3,1-ch[i]}^i)$ is invalid. The probability (over the choice of ch) of such an event happening is $2^{-\lambda}$.

Notice that the only difference between the real world and the ideal world distribution is in generating the garbled circuit $\widetilde{C}$. In the real world, it is generated honestly whereas in the ideal simulation the garbled circuit is simulated. We note that the output μ used in the generation of $\widetilde{C}$ is same as the real world output. It now follows from the security of the garbled circuits that the real world and ideal world distributions are computationally indistinguishable. ■

5.2.3 Case-3: Neither P_1 nor P_2 is corrupted

We describe the ideal world simulator in Figure 13.

Common Random String: Sample $s \leftarrow \{0,1\}^\lambda$ and set $\sigma := \mathsf{PRG}(s)$ as the CRS.

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

- Receive $(R(x,w), x, \mu)$ from the ideal functionality $\mathcal{F}_{\mathsf{ZK}}$.
- **Message from Sim on behalf of P_1:** Sim does the following:
  1. **if**$(R(x,w) = 0)$, send $(\mathsf{NotInL}, x, 0)$ to every party. Else, do the following.
  2. **for** each $i \in [\lambda]$ **do**:
     (a) Prepare zk_1^i for the language $\mathcal{L}$ using the witness s.
     (b) Let $\mathsf{zk}_{3,b}^i$ be the third round message when $\mathsf{zk}_2 = b$. Sample $r_b^i \leftarrow \{0,1\}^\lambda$ for each $b \in \{0,1\}$ and compute $c_b^i := \mathsf{com}(\mathsf{zk}_{3,b}^i; r_b^i)$.
     (c) Broadcast $\mathsf{zk}_1^i, c_0^i, c_1^i$ to every other party.
- **Message from Sim on behalf of P_2:** Sim does the following:
  1. **if**$(R(x,w) = 0)$, send μ to every party. Else, do the following.
  2. Sample $\mathsf{lab}_w^i \leftarrow \{0,1\}^\lambda$ for each $i, w \in [\lambda]$.
  3. Compute $\widetilde{C} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, 1^{|C|}, \mu, \{\mathsf{lab}_w^i\}_{i,w \in \mathsf{inp}(C)}\right)$
  4. Broadcast $\widetilde{C}$ and send $\{\mathsf{lab}_w^i\}_{i,w \in [\lambda]}$ as the output from $\mathcal{F}_f$ to every party.

Figure 13: Simulator Sim when neither P_1 nor P_2 is corrupted

It follows directly from cases-1 and 2 that the real world distribution is computationally indistinguishable from ideal world distribution.

6 Malicious Secure Protocol

In this section, we give a construction of two-round, multiparty computation that is secure against malicious adversaries and minimizes the number of public key operations.

6.1 Construction

Our two-round protocol computing a function f uses the following primitives.

1. An n-party malicious secure conforming protocol Φ computing the function f.
2. A selective garbling scheme for circuits $(\mathsf{Garble}, \mathsf{Eval})$.
3. A pseudorandom generator $\mathsf{PRG}_{\mathsf{mal}} : \{0,1\}^\lambda \to \{0,1\}^{4T}$ where each output bit can be computed by a circuit of size $\mathsf{poly}(\lambda, \log T)$.[6]
4. A somewhere adaptive garbling scheme for circuits $(\mathsf{SAdpGarbleCkt}, \mathsf{SAdpGarbleInp}, \mathsf{SAdpEvalCkt})$ (defined in Section 3.2). We assume that the length of the garbled input when $\mathsf{SAdpGarbleCkt}$ is used to garbled $\mathsf{C_B}$ (described in Figure 14) is M.
5. A maliciously secure two-round MPC protocol computing the function g described in Figure 15.
6. A non-interactive statistically binding commitment scheme $(\mathsf{Com}, \mathsf{Decom})$.
7. The special ZK protocol parameterized by an NP relation R described below.

$$R \ := \ \left\{ \left(x = \left(\widetilde{\mathsf{C}}_{\mathsf{B}}, \mathsf{cm} \right), w = (\Omega, \mathsf{C_B}, \mathsf{state}, \omega) \right) : \right.$$
$$\left. (\mathsf{Decom}(\mathsf{cm}, \mathsf{state}, \omega) = 1) \wedge \left((\widetilde{\mathsf{C}}_{\mathsf{B}}, \mathsf{state}) = \mathsf{SAdpGarbleCkt}\left(\mathsf{C_B}; \Omega\right) \right) \right\}.$$

We give an overview of the construction below and describe the formal construction later.

Overview. In order to make our protocol maliciously secure, we need to make the special purpose OT extension protocol maliciously secure. Below, we will first describe the changes between a single pair of sender and receiver, and then extending it to $n-1$ senders with a single receiver is similar to the semi-honest setting.

In the first round, the message sent from the receiver is the same as the semi-honest setting. The sender samples a set of masks $M = \{\mathsf{m}_{i,0}, \mathsf{m}_{i,1}\}_{i \in [L]}$ and constructs $\mathsf{C_B}$ same as before. It then uses the somewhere adaptive garbling scheme to garble $\mathsf{C_B}$ using randomness Ω and obtains $\widetilde{\mathsf{C}}_{\mathsf{B}}$ and a secret state state. The sender sends $\widetilde{\mathsf{C}}_{\mathsf{B}}$ to the receiver. Additionally, it computes a commitment cm of state using randomness ω.

In the second round, the sender computes and hardwires $\{ct_{i,b}\}_{i \in [L], b \in \{0,1\}}$ in its "talking garbled circuits" same as before. The receiver constructs a wrap-circuit $\mathsf{C_{wrap}}$ that is the same as before

[6] The GGM PRF [GGM86] can be easily modified to give such a PRG based on one-way functions.

except that it takes an additional bit b as input indicating whether $\widetilde{\mathsf{C}}_{\mathsf{B}}$ is constructed correctly by the sender. The receiver then garbles $\mathsf{C}_{\mathsf{wrap}}$ and sends the garbled circuit $\widetilde{\mathsf{C}}_{\mathsf{wrap}}$ to the sender.

In order to obtain the label for the additional input bit b of $\widetilde{\mathsf{C}}_{\mathsf{B}}$, the sender and receiver in parallel run the two-round special zero-knowledge protocol for the language R defined above. Specially, $\mathcal{F}_{\mathsf{zk}}$ takes as input $\left(x = \left(\widetilde{\mathsf{C}}_{\mathsf{B}}, \mathsf{cm}\right), w = (\Omega, \mathsf{C}_{\mathsf{B}}, \mathsf{state}, \omega)\right)$ from the sender and $(\mathsf{lab}_0, \mathsf{lab}_1)$ from the receiver, where $(\mathsf{lab}_0, \mathsf{lab}_1)$ are the labels for the input bit b. If $\widetilde{\mathsf{C}}_{\mathsf{B}}$ is generated correctly and ω is the randomness used to generate cm, then $\mathcal{F}_{\mathsf{zk}}$ will output lab_1 for evaluating $\widetilde{\mathsf{C}}_{\mathsf{wrap}}$.

In order to obtain the remaining labels for evaluating $\widetilde{\mathsf{C}}_{\mathsf{wrap}}$, the sender and receiver in parallel run a two-round maliciously secure MPC protocol from [GS18] to compute a function g. The function g is different from before in that it takes state as input to compute the garbled input of $\widetilde{\mathsf{C}}_{\mathsf{B}}$. In addition, it first checks if the sender has the randomness used to generate cm. If so, then it computes the two-step translation functionality and outputs the labels for evaluating $\widetilde{\mathsf{C}}_{\mathsf{wrap}}$. Notice that the statistically binding commitment scheme ensures that state fed into $\mathcal{F}_{\mathsf{zk}}$ is consistent with that fed into $\mathcal{F}_g$ with overwhelming probability.

At the end of the second round, the two parties will obtain lab_1 along with all the remaining labels for evaluating $\widetilde{\mathsf{C}}_{\mathsf{wrap}}$ only if the sender has generated $\widetilde{\mathsf{C}}_{\mathsf{B}}$ correctly. Hence they are able to evaluate $\widetilde{\mathsf{C}}_{\mathsf{wrap}}$ at the end of the second round.

The remaining steps in constructing our protocol are (a) extending the maliciously secure special purpose OT extension protocol between a single pair of sender and receiver to a protocol with $n-1$ senders and a single receiver, and (b) combine the maliciously secure special purpose OT extension with the maliciously secure MPC protocol of [GS18]. These steps are similar to the semi-honest setting.

Description of the Protocol. We now give a formal description of our construction in below in the $\mathcal{F}_g$ and $\mathcal{F}_{\mathsf{zk}}$ hybrid model.

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 $t \in [T]$ and for each $\alpha, \beta \in \{0,1\}$
$$c_i[(t,\alpha,\beta)] := v_i[h] \oplus \mathsf{NAND}(v_i[f] \oplus \alpha, v_i[g] \oplus \beta)$$
where $\phi_t = (\star, f, g, h)$.
3. Sample $s_i \leftarrow \{0,1\}^\lambda$ and compute $e_i := \mathsf{PRG}_{\mathsf{mal}}(s_i) \oplus c_i$.
4. For each $j \in [n] \setminus \{i\}$, sample
$$\begin{aligned} \mu_0^{j\to i}, \mu_1^{j\to i} &\leftarrow \{0,1\}^\lambda \\ \mathsf{rlab}_{k,b}^{j\to i} &\leftarrow \{0,1\}^\lambda \text{ for all } k \in [M], b \in \{0,1\} \\ \mathsf{m}_{k,b}^{i\to j} &\leftarrow \{0,1\}^\lambda \text{ for all } k \in [4T], b \in \{0,1\} \end{aligned}$$
5. **Garbling C_{B}:** For each $j \in [n] \setminus \{i\}$, compute
$$\begin{aligned} (\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}, \mathsf{state}^{i\to j}) &:= \mathsf{SAdpGarbleCkt}\left(\mathsf{C}_{\mathsf{B}}\left[\left\{\mathsf{m}_{k,0}^{i\to j}, \mathsf{m}_{k,1}^{i\to j}\right\}_{k\in[4T], b\in\{0,1\}}\right]; \Omega\right) \\ \mathsf{cm}^{i\to j} &:= \mathsf{Com}(\mathsf{state}^{i\to j}; \omega^{i\to j}) \end{aligned}$$

where $\mathsf{C_B}$ is described in Figure 14 and $\Omega, \omega^{i\to j}$ are sampled randomly.

6. **Messages to $\mathcal{F}_g$:** Send $(\mathsf{ssid} = i, s_i, \{\mathsf{rlab}_{k,b}^{j\to i}\}_{j\in[n]\setminus\{i\},k\in[M],b\in\{0,1\}})$ to $\mathcal{F}_g$ acting as the receiver and for each $j \in [n] \setminus \{i\}$, send $(\mathsf{ssid} = j, \{\mathsf{cm}^{i\to j}, \mathsf{state}^{i\to j}, \omega^{i\to j}\})$ to $\mathcal{F}_g$ acting as the sender.
7. **Messages to $\mathcal{F}_{\mathsf{zk}}$:** For each $j \in [n] \setminus \{i\}$, send $(\mathsf{ssid} = (j \to i), \mu_0^{j\to i}, \mu_1^{j\to i})$ to $\mathcal{F}_{\mathsf{zk}}$ acting as the verifier, and send $(\mathsf{ssid} = (i \to j), X^{i\to j}, W^{i\to j})$ to $\mathcal{F}_{\mathsf{zk}}$ acting as the prover where $X^{i\to j} = \left(\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}, \mathsf{cm}^{i\to j}\right)$ and $W^{i\to j} = \left(\Omega, \mathsf{C_B}\left[\left\{\mathsf{m}_{k,0}^{i\to j}, \mathsf{m}_{k,1}^{i\to j}\right\}_{k\in[4T],b\in\{0,1\}}\right], \mathsf{state}^{i\to j}, \omega^{i\to j}\right)$.
8. Send $\left(z_i, \{\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}\}_{j\in[n]\setminus\{i\}}, e_i, \{\mathsf{cm}^{i\to j}\}_{j\in[n]\setminus\{i\}}\right)$ to every other party.

$$\mathsf{C_B}\left[\{\mathsf{m}_{k,0}, \mathsf{m}_{k,1}\}_{k\in[4T]}\right]$$

Input: $s \in \{0,1\}^\lambda$.

1. $d := \mathsf{PRG_{mal}}(s)$ where $d \in \{0,1\}^{4T}$.
2. Output $\left\{\mathsf{m}_{k,d[k]}\right\}_{k\in[4T]}$.

Figure 14: Circuit $\mathsf{C_B}$

Parties: $P_1, P_2, \ldots, P_n$.
Inputs:

- P_1 (also called as the receiver) inputs $s \in \{0,1\}^\lambda$ and $\overline{\mathsf{rlab}}_2, \ldots, \overline{\mathsf{rlab}}_n$ where each $\overline{\mathsf{rlab}}_i$ is a collection of labels $\{\mathsf{rlab}_{j,0}^{i\to 1}, \mathsf{rlab}_{j,1}^{i\to 1}\}_{j\in[M]}$ with each label of length λ.
- For each $i \in [2,n]$, P_i (also called as the sender) inputs $(\mathsf{cm}^{i\to 1}, \mathsf{state}^{i\to 1}, \omega^{i\to 1})$, where $\mathsf{cm}^{i\to 1}$ is a commitment and is a public input, $\mathsf{state}^{i\to 1}$ is the secret state of the somewhere adaptive garbling scheme, and $\omega^{i\to 1}$ is a string.

Output: Check if for each $i \in [2,n]$, $\mathsf{Decom}(\mathsf{cm}^{i\to 1}, \mathsf{state}^{i\to 1}, \omega^{i\to 1}) = 1$. If all the checks pass, output $\{\mathsf{Projection}(\mathsf{SAdpGarbleInp}(\mathsf{state}^{i\to 1}, s), \overline{\mathsf{rlab}}_i)\}_{i\in[2,n]}$ to every party.

Figure 15: The function g computed by the internal MPC where P_1 acts as the receiver

Round-2: Each party P_i does the following:

1. Set $\mathsf{st}_i := (z_1\|\ldots\|z_n) \oplus v_i$.
2. Set $N = \ell + 4T\lambda(n-1)$.
3. Set $\overline{\mathsf{lab}}^{i,T+1} := \{\mathsf{lab}_{k,0}^{i,T+1}, \mathsf{lab}_{k,1}^{i,T+1}\}_{k\in[N]}$ where $\mathsf{lab}_{k,b}^{i,T+1} := 0^\lambda$ for each $k \in [N], b \in \{0,1\}$.
4. **for** each t from T down to 1 **do**:
   (a) Parse ϕ_t as (i^*, f, g, h).

(b) If $i = i^*$ then compute (where P is described in Figure 6)

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

(c) If $i \neq i^*$ then for every $\alpha, \beta \in \{0,1\}$, set $\mathsf{m}'_{\alpha,\beta,0} = \mathsf{m}^{i\to i^*}_{(t,\alpha,\beta),e_{i^*}[(t,\alpha,\beta)]}$ and $\mathsf{m}'_{\alpha,\beta,1} = \mathsf{m}^{i\to i^*}_{(t,\alpha,\beta),1\oplus e_{i^*}[(t,\alpha,\beta)]}$.
Compute $ct^i_{t,\alpha,\beta} := (\mathsf{m}'_{\alpha,\beta,0} \oplus \mathsf{lab}^{i,t+1}_{h,0}, \mathsf{m}'_{\alpha,\beta,1} \oplus \mathsf{lab}^{i,t+1}_{h,1})$ and compute

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

5. **Garbling $\mathsf{C}_{\mathsf{wrap}}$:** Compute

$$\widetilde{\mathsf{C}}^i_{\mathsf{wrap}} \leftarrow \mathsf{Garble}\Big(\mathsf{C}_{\mathsf{wrap}}\left[\{\widetilde{\mathsf{C}}^{j\to i}_{\mathsf{B}}\}_{j\in[n]\setminus\{i\}}, \mathsf{st}_i, \overline{\mathsf{lab}}^{i,1}\right], \left\{\{\mu^{j\to i}_b\}_{j\in[n]\setminus\{i\}}, \{\mathsf{rlab}^{j\to i}_{k,b}\}_{j\in[n]\setminus\{i\},k\in[M],b\in\{0,1\}}\right\}\Big)$$

where $\mathsf{C}_{\mathsf{wrap}}$ is described in Figure 16.

6. Send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \widetilde{\mathsf{C}}^i_{\mathsf{wrap}}\right)$ to every other party.

$$\mathsf{C}_{\mathsf{wrap}}\left[\{\widetilde{\mathsf{C}}^{j\to i}_{\mathsf{B}}\}_{j\in[n]\setminus\{i\}}, \mathsf{st}_i, \overline{\mathsf{lab}}^{i,1}\right]$$

Input: $\{b^{j\to i}\}_{j\in[n]\setminus\{i\}}, \{\widetilde{s}^{j\to i}\}_{j\in[n]\setminus\{i\}}$

1. If $b^{j\to i} = 1$ for all $j \in [n] \setminus \{i\}$ do:
   (a) For each $j \in [n] \setminus \{i\}$, compute $\left\{\mathsf{m}^{j\to i}_k\right\}_{k\in[4T]} \leftarrow \mathsf{SAdpEvalCkt}\left(\widetilde{\mathsf{C}}^{j\to i}_{\mathsf{B}}, \widetilde{s}^{j\to i}\right)$.
   (b) Let $\mathsf{m} := \underset{j\in[n]\setminus\{i\},k\in[4T]}{\|}(\mathsf{m}^{j\to i}_k)$
   (c) Output $\mathsf{Projection}(\mathsf{st}_i \| \mathsf{m}, \overline{\mathsf{lab}}^{i,1})$.
2. Else, output $\perp$.

Figure 16: Circuit $\mathsf{C}^{\mathsf{wrap}}$

Evaluation: Every party P_i does the following:

1. For each $j \in [n]$,
   (a) Obtain ($\mathsf{ssid} = j, \{\widetilde{\mathsf{rlab}}^j\}$) from $\mathcal{F}_g$ where party P_j acts as the receiver.
   (b) For each $k \in [n] \setminus \{j\}$, obtain ($\mathsf{ssid} = (k \to j), b^{k\to j}, X^{k\to j}, \mu^{k\to j}$) from $\mathcal{F}_{\mathsf{zk}}$. Set $\widetilde{\mu}^j = \{\mu^{k\to j}\}_{k\in[n]\setminus\{j\}}$.
   (c) $\widetilde{\mathsf{lab}}^{j,1} \leftarrow \mathsf{Eval}(\widetilde{\mathsf{C}}^j_{\mathsf{wrap}}, \widetilde{\mu}^j \| \widetilde{\mathsf{rlab}}^j)$.
2. **for** each t from 1 to T **do**:
   (a) Parse ϕ_t as (i^*, f, g, h).

   (b) Compute $((\alpha, \beta, \gamma), \{\omega^j\}_{j \in [n] \setminus \{i\}}, \widetilde{\mathsf{lab}}^{i^*,t+1}) := \mathsf{Eval}(\widetilde{\mathsf{P}}^{i^*,t}, \widetilde{\mathsf{lab}}^{i^*,t})$.
   (c) Set $\mathsf{st}_i[h] := \gamma \oplus v_i[h]$.
   (d) **for** each $j \neq i^*$ **do**:
      i. Compute $(ct = (\delta_0, \delta_1), \{\mathsf{lab}_k^{j,t+1}\}_{k \in [N] \setminus \{h\}}) := \mathsf{Eval}(\widetilde{\mathsf{P}}^{j,t}, \widetilde{\mathsf{lab}}^{j,t})$.
      ii. Recover $\mathsf{lab}_h^{j,t+1} := \delta_\gamma \oplus \omega^j$.
      iii. Set $\widetilde{\mathsf{lab}}^{j,t+1} := \{\mathsf{lab}_k^{j,t+1}\}_{k \in [N]}$.
3. Compute the output as $\mathsf{post}(i, \mathsf{st}_i)$.

Correctness. The correctness follows via a similar argument to the semi-honest case.

Efficiency. Let the number of public key operations in Φ be npk_Φ, in one execution of $\mathcal{F}_{\mathsf{zk}}$ be $\mathsf{npk}_{\mathsf{zk}}$, and in one execution of $\mathcal{F}_g$ be npk_g. We choose the conforming protocol that performs OT extension between every pair of parties so that npk_Φ is bounded by $\mathcal{O}(n^2\lambda)$. The total number of public key operations in our two-round construction is $\mathcal{O}(\mathsf{npk}_\Phi + n^2 \cdot \mathsf{npk}_{\mathsf{zk}} + n \cdot \mathsf{npk}_g)$. It follows from Theorems 5.2, 3.3 that this number is bounded by $\mathsf{poly}(n, \lambda)$.

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 from [GS18] and then describe our simulator.

Faithful Execution [GS18]. In the first round of our compiled protocol, $\mathcal{A}$ provides z_i for every $i \in [n] \setminus H$ and $c_i \oplus \mathsf{PRG}_{\mathsf{mal}}(s_i)$ and sends $(s_i, \{\mathsf{rlab}_{k,b}^{j \to i}\})$ to the $\mathcal{F}_g$ functionality. These messages act as "binding" commitments to the value c_i which in turn determines all of the adversary's future choices. A simulator can extract the value c_i by observing these messages. Intuitively speaking, a faithful execution is an execution that is consistent with these extracted values.

More formally, we define an interactive procedure $\mathsf{Faithful}(i, \{z_j\}_{j \in [n]}, c_i)$ that on input $i \in [n]$, $\{z_j\}_{j \in [n]}$, c_i, produces protocol Φ messages 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 $\mathsf{st}^*[h]$ to be the received bit once it is received.
   (c) Set $\mathsf{st}^*[h] := c_i[(t, \mathsf{st}^*[f], \mathsf{st}^*[g])]$ 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 S that simulates the view of the real world adversary $\mathcal{A}$. S will internally use the malicious simulator Sim_Φ for Φ and the simulator $\mathsf{Sim}_{\mathsf{ckt}}$ for selective garbling of circuits.

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

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

- **Initialization**: S executes the simulator $\mathsf{Sim}_\Phi(1^\lambda)$ to obtain $\{z_i\}_{i\in H}$. Moreover, S starts the real-world adversary $\mathcal{A}$.
- **Round-1 messages from S to $\mathcal{A}$:** For each $i \in H$, S generates the first round message on behalf of P_i as follows:
  - Sample $e_i \leftarrow \{0,1\}^{4T}$.
  - For each $j \in [n] \setminus \{i\}$, sample
  $$\mathsf{m}_k^{i\to j} \leftarrow \{0,1\}^\lambda \text{ for all } k \in [4T].$$
  - For each $j \in [n] \setminus \{i\}$, compute
  $$\begin{aligned}(\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}, \mathsf{state}^{i\to j}) &\leftarrow \mathsf{SAdpGarbleCkt}\left(\mathsf{C}_{\mathsf{D}}\left[\left\{\mathsf{m}_k^{i\to j}\right\}_{k\in[4T]}\right]\right)\\ \mathsf{cm}^{i\to j} &\leftarrow \mathsf{Com}(\vec{0})\end{aligned}$$
  where $\mathsf{C}_{\mathsf{D}}\left[\left\{\mathsf{m}_k^{i\to j}\right\}_{k\in[4T]}\right]$ is a dummy circuit that outputs $\{\mathsf{m}_k^{i\to j}\}_{k\in[4T]}$ as output on every input.
  - Send $\left(z_i, \{\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}\}_{j\in[n]\setminus\{i\}}, e_i, \{\mathsf{cm}^{i\to j}\}_{j\in[n]\setminus\{i\}}\right)$ to every other party.
- **Round-1 messages from $\mathcal{A}$ to S:** Corresponding to every $i \in [n] \setminus H$, S receives from the adversary $\mathcal{A}$ the value $(z_i, \{\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}\}_{j\in[n]\setminus\{i\}}, e_i, \{\mathsf{cm}^{i\to j}\}_{j\in[n]\setminus\{i\}})$ on behalf of the corrupted party P_i. In addition to these,
  - S receives the values $(\mathsf{ssid} = i, s_i, \{\mathsf{rlab}_{k,b}^{j\to i}\})$ that $\mathcal{A}$ sends to the $\mathcal{F}_g$ functionality. Using s_i and e_i, S computes $c_i = \mathsf{PRG}_{\mathsf{mal}}(s_i) \oplus e_i$. It also receives $(\mathsf{ssid} = j, \{\mathsf{cm}^{i\to j}, \mathsf{state}^{i\to j}, \omega^{i\to j}\})$ sent to $\mathcal{F}_g$ for every $j \in [n] \setminus \{i\}$. It stores all these values.
  - For each $j \in [n] \setminus \{i\}$, S receives $(\mathsf{ssid} = (j \to i), \mu_0^{j\to i}, \mu_1^{j\to i})$ that $\mathcal{A}$ sends to the $\mathcal{F}_{\mathsf{zk}}$ acting as the verifier. It also receives $(\mathsf{ssid} = (i \to j), X^{i\to j}, W^{i\to j})$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{zk}}$ acting as the prover. It again stores all the values.
- **Completing the execution with the Sim_Φ:** For each $i \in [n] \setminus H$, S sends z_i to Sim_Φ on behalf of the corrupted party P_i. This starts the computation phase of Φ with the simulator Sim_Φ. S provides computation phase messages to Sim_Φ by following a faithful execution.

More formally, for every corrupted party P_i where $i \in [n] \setminus H$, S generates messages on behalf of P_i for Sim_Φ using the procedure $\mathsf{Faithful}(i, \{z_j\}_{j\in[n]}, c_i)$. At some point during the execution, Sim_Φ will return the extracted inputs $\{x_i\}_{i\in[n]\setminus H}$ of the corrupted parties. For each $i \in [n] \setminus H$, S sends $(\mathsf{input}, \mathsf{sid}, \{P_1 \cdots P_n\}, P_i, x_i)$ to the ideal functionality implementing f and obtains the output y which is provided to Sim_Φ. Finally, at some point the faithful execution completes.

Let $\mathsf{Z} \in \{0,1\}^T$ be output of this execution where $\mathsf{Z}[t]$ is the bit sent in the t-th round of the computation phase of Φ. 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 corrupted parties).

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

  1. For each $j \in [n] \setminus H$, S recovers $(\mathsf{ssid} = (j \to i), X^{j\to i}, W^{j\to i})$ from its storage. It sets $b^{j\to i} := R(X^{j\to i}, W^{j\to i})$. For all $j \in H \setminus \{i\}$, it sets $b^{j\to i} := 1$.
  2. For each $j \in [n] \setminus \{i\}$, sample
  $$\begin{aligned} \mu^{j\to i} &\leftarrow \{0,1\}^\lambda \\ \mathsf{rlab}_k^{j\to i} &\leftarrow \{0,1\}^\lambda \text{ for all } k \in [M] \end{aligned}$$
  3. If for any $j \in [n] \setminus H$, $b^{j\to i} = 0$, then:
     (a) **for** each t from T down to 1,
        i. For each $k \in [N]$, sample $\mathsf{lab}_k^{i,t} \leftarrow \{0,1\}^\lambda$.
        ii. Compute
        $$\widetilde{\mathsf{P}}^{i,t} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, (\bot), \{\mathsf{lab}_k^{i,t}\}_{k\in[N]}\right).$$
     (b) Compute
     $$\widetilde{\mathsf{C}}^i_{\mathsf{wrap}} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, (\bot), \left\{\{\mu^{j\to i}\}_{j\in[n]\setminus\{i\}}, \{\mathsf{rlab}_k^{j\to i}\}_{j\in[n]\setminus\{i\},k\in[M]}\right\}\right).$$
     (c) Send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \widetilde{\mathsf{C}}^i_{\mathsf{wrap}}\right)$ to every other party.
  4. Otherwise:
     (a) For each $j \in [n] \setminus \{H \cup \{i\}\}$, recover $\mathsf{C_B}\left[\left\{\mathsf{m}_{k,0}^{j\to i}, \mathsf{m}_{k,1}^{j\to i}\right\}_{k\in[4T]}\right]$ from $W^{j\to i}$. S stores $\left\{\mathsf{m}_{k,0}^{j\to i}, \mathsf{m}_{k,1}^{j\to i}\right\}_{k\in[4T]}$.
     (b) For each $k \in [N]$, set $\mathsf{lab}_k^{i,T+1} := 0^\lambda$.
     (c) **for** each t from T down to 1,
        i. Parse ϕ_t as (i^*, f, g, h).
        ii. Set $\alpha^* := \mathsf{st}^*[f]$, $\beta^* := \mathsf{st}^*[g]$, and $\gamma^* := \mathsf{st}^*[h]$.
        iii. For each $k \in [N]$, sample $\mathsf{lab}_k^{i,t} \leftarrow \{0,1\}^\lambda$.
        iv. If $i = i^*$ then compute
        $$\left(\widetilde{\mathsf{P}}^{i,t}\right) \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, \left((\alpha^*, \beta^*, \gamma^*), \omega_{t,\alpha^*,\beta^*}, \{\mathsf{lab}_k^{i,t+1}\}_{k\in[N]}\right), \{\mathsf{lab}_k^{i,t}\}_{k\in[N]}\right).$$
        where $\omega_{t,\alpha^*,\beta^*} = \{\mathsf{m}^{j\to i}_{(t,\alpha^*,\beta^*),\mathsf{Z}[t]\oplus e_i[(t,\alpha^*,\beta^*)]}\}_{j\in[n]\setminus\{i\}}$.

v. If $i \neq i^*$ then set $\delta_{\mathsf{Z}[t]} := \mathsf{m}^{i \to i^*}_{(t,\alpha^*,\beta^*)} \oplus \mathsf{lab}^{i,t+1}_h$ and $\delta_{1-\mathsf{Z}[t]} \leftarrow \{0,1\}^\lambda$. Set $ct := (\delta_0, \delta_1)$ and compute

$$\widetilde{\mathsf{P}}^{i,t} \leftarrow \mathsf{Sim}_{\mathsf{ckt}} \left(1^\lambda, \left(ct, \{\mathsf{lab}^{i,t+1}_k\}_{k \in [N] \setminus \{h\}}\right), \{\mathsf{lab}^{i,t}_k\}_{k \in [N]}\right).$$

(d) Compute

$$\widetilde{\mathsf{C}}^i_{\mathsf{wrap}} \leftarrow \mathsf{Sim}_{\mathsf{ckt}} \left(1^\lambda, (\{\mathsf{lab}^{i,1}_k\}_{k \in [N]}), \left\{\{\mu^{j \to i}\}_{j \in [n] \setminus \{i\}}, \{\mathsf{rlab}^j_k\}_{j \in [n] \setminus \{i\}, k \in [M]}\right\}\right).$$

(e) Send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t \in [T]}, \widetilde{\mathsf{C}}^{\mathsf{wrap}}_i\right)$ to every other party.

- **Output from $\mathcal{F}_g$ functionality.** For each $i \in H$, S sets the output from $\mathcal{F}_g$ where P_i acts as the receiver to be $\{\mathsf{rlab}^{j \to i}_k\}_{j \in [n] \setminus \{i\}, k \in [N]}$. If $i \notin H$, it computes the output of the $\mathcal{F}_g$ functionality using $\{\mathsf{state}^{j \to i}\}_{j \in [n] \setminus \{i\}}$. That is, it uses the $\mathsf{state}^{j \to i}$ to compute $\widetilde{s}^{j \to i}$ and then uses it to compute the output as $\mathcal{F}_g$.

- **Output from $\mathcal{F}_{\mathsf{zk}}$ functionality.** For each $(i, j) \in [N]$, if $j \in H$ then S sets the output from $\mathcal{F}_{\mathsf{zk}}$ of $\mathsf{ssid} = (i \to j)$ as $\mu^{i \to j}$. If $j \notin H$ and $i \in H$, it sets the output to be $\mu^{i \to j}_1$. Else, if both $i, j \notin H$, it acts exactly as $\mathcal{F}_{\mathsf{zk}}$.

- **Round-2 messages from $\mathcal{A}$ to S:** For every $i \in [n] \setminus H$, S obtains the second round message from $\mathcal{A}$ on behalf of the malicious parties. Subsequent to obtaining these messages, S executes the garbled circuits provided by $\mathcal{A}$ on behalf of the corrupted parties to see the execution of garbled circuits proceeds consistently with the expected faithful execution. If the computation succeeds then for each $i \in H$, 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 S.

- $\mathcal{H}_0$: This hybrid is the same as the real world execution.

- $\mathcal{H}_1$: In this hybrid, the functionalities $\mathcal{F}_g$ and $\mathcal{F}_{\mathsf{zk}}$ are emulated by the simulator S faithfully. This hybrid is identical to $\mathcal{H}_0$.

- $\mathcal{H}_2$: In this hybrid, we change how the $\mathcal{F}_{\mathsf{zk}}$ functionality is emulated by the simulator in every call where $i \in H$ is acting as the prover. In particular, for any $\mathsf{ssid} = (i \to j)$ where $i \in H$, the party P_i does not send $(X^{i \to j}, W^{i \to j})$ to the ideal functionality and the simulator always sets the output to be $(1, X^{i \to j}, \mu^{i \to j}_1)$. This hybrid is identically distributed to $\mathcal{H}_1$.

- $\mathcal{H}_3$: In this hybrid, we change how the $\mathcal{F}_g$ functionality is emulated by the simulator in every call where $i \in H$ is acting as the sender. In particular, for any $\mathsf{ssid} = j \in [n]$, when $i \in H$ is acting as the sender in this execution, party P_i does not send $(\mathsf{cm}^{i \to j}, \mathsf{state}^{i \to j}, \omega^{i \to j})$ to the ideal functionality. Instead, the simulator uses $\mathsf{state}^{i \to j}$ to directly compute the output. This hybrid is distributed identically to $\mathcal{H}_2$.

- $\mathcal{H}_4$: In this hybrid, we change $\mathsf{cm}^{i\to j}$ for every $i \in H$ and $j \in [n] \setminus \{i\}$ to be $\mathsf{Com}(\vec{0})$ instead of $\mathsf{Com}(\mathsf{state}^{i\to j})$. $\mathcal{H}_4$ is computationally indistinguishable from $\mathcal{H}_3$ from the computational hiding property of the non-interactive commitment scheme.
- $\mathcal{H}_5$: In this hybrid, we make two changes with respect to $\mathcal{H}_4$ which are explained below. For every $i \in H$:
  1. In round-1, for each $j \in [n] \setminus \{i\}$, we sample
  $$\mathsf{m}_k^{i\to j} \leftarrow \{0,1\}^\lambda \text{ for all } k \in [4T]$$
  and compute
  $$(\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}, \mathsf{state}^{i\to j}) \leftarrow \mathsf{SAdpGarbleCkt}\left(\mathsf{C}_{\mathsf{D}}\left[\left\{\mathsf{m}_k^{i\to j}\right\}_{k\in[4T]}\right]\right).$$
  2. At the end of round-1, we obtain s_j for each $j \in [n] \setminus \{i\}$. For those $j \in [n] \setminus \{H\}$ this is obtained from the message that the adversary sends to $\mathcal{F}_g$. For the rest, we obtain it directly using the random coins of the honest parties. In round-2, for each $j \in [n] \setminus \{i\}$ and $k \in [4T]$, we set $\mathsf{m}^{i\to j}_{k,\mathsf{PRG}_{\mathsf{mal}}(s_j)[k]} = \mathsf{m}_k^{i\to j}$ and $\mathsf{m}^{i\to j}_{k,1-\mathsf{PRG}_{\mathsf{mal}}(s_j)[k]} \leftarrow \{0,1\}^\lambda$. We use these values to compute $\{ct^i_{t,\alpha,\beta}\}$.

We show that $\mathcal{H}_4$ is computationally indistinguishable to $\mathcal{H}_5$ using a sequence of sub-hybrids. We show how to do this change for any $i \in H$ and the change to every $i \in H$ can be obtained by repeating this hybrid sequence $|H|$ times separately for each party in H.

Notation. We introduce a total ordering $\prec$ of the tuples (t,α,β) where $t \in [T], \alpha,\beta \in \{0,1\}$. We say that $(t,\alpha,\beta) \prec (t',\alpha',\beta')$ if $t < t'$ or if $t = t'$ and $\alpha < \alpha'$ or if $t = t'$, $\alpha = \alpha'$ and $\beta < \beta'$. For each $t \in [0,T]$, $\alpha,\beta \in \{0,1\}$, we define a hybrid,

- $\mathcal{H}_{4,(t,\alpha,\beta)}$: Let $K := \{(t',\alpha',\beta') : (t',\alpha',\beta') \preceq (t,\alpha,\beta)\}$. We generate $\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}$ as follows:
  1. In round-1, for each $j \in [n] \setminus \{i\}$, sample
  $$\begin{aligned} \mathsf{m}_k^{i\to j} &\leftarrow \{0,1\}^\lambda \text{ for all } k \in K \\ \mathsf{m}_{k,0}^{i\to j}, \mathsf{m}_{k,1}^{i\to j} &\leftarrow \{0,1\}^\lambda \text{ for all } k \in [4T] \setminus K \end{aligned}$$
  and compute,
  $$(\widetilde{\mathsf{C}}_{\mathsf{B}}^{i\to j}, \mathsf{state}^{i\to j}) \leftarrow \mathsf{SAdpGarbleCkt}\left(\mathsf{C}_{\mathsf{inter}}\left[\left\{\mathsf{m}_k^{i\to j}\right\}_{k\in K}, \left\{\mathsf{m}_{k,0}^{i\to j}, \mathsf{m}_{k,1}^{i\to j}\right\}_{k\in[4T]\setminus K}\right]\right)$$
  where $\mathsf{C}_{\mathsf{inter}}$ is described in Figure 17.
  2. In round-2, for each $k \in K$ and $j \in [n] \setminus \{i\}$, set $\mathsf{m}^{i\to j}_{k,\mathsf{PRG}_{\mathsf{mal}}(s_j)[k]} = \mathsf{m}_k^{i\to j}$ and $\mathsf{m}^{i\to j}_{k,1-\mathsf{PRG}_{\mathsf{mal}}(s_j)[k]} \leftarrow \{0,1\}^\lambda$.

Note that $\mathcal{H}_4$ is distributed identically to $\mathcal{H}_{4,(0,0,0)}$ and $\mathcal{H}_5$ is distributed identically to $\mathcal{H}_{4,(T,1,1)}$. Thus, to prove indistinguishability between $\mathcal{H}_4$ and $\mathcal{H}_5$, it is sufficient to show indistinguishability between $\mathcal{H}_{4,(t',\alpha',\beta')}$ and $\mathcal{H}_{4,(t,\alpha,\beta)}$ for any two adjacent (induced by the ordering $\prec$) (t',α',β') and (t,α,β). We argue this indistinguishability using somewhere adaptive security of $(\mathsf{SAdpGarbleCkt}, \mathsf{SAdpGarbleInp})$.

$$\mathsf{C}_{\mathsf{inter}}\left[\left\{\mathsf{m}_k^{i\to j}\right\}_{k\in K},\left\{\mathsf{m}_{k,0}^{i\to j},\mathsf{m}_{k,1}^{i\to j}\right\}_{k\in[4T]\setminus K,b\in\{0,1\}}\right]$$

Input: $s \in \{0,1\}^\lambda$.

1. $d := \mathsf{PRG}_{\mathsf{mal}}(s)$ where $d \in \{0,1\}^{4T}$.
2. Output $\left\{\mathsf{m}_k^{i\to j}\right\}_{k\in K},\left\{\mathsf{m}_{k,d[k]}^{i\to j}\right\}_{k\in[4T]\setminus K}$.

Figure 17: Circuit $\mathsf{C}_{\mathsf{inter}}$

Claim 6.1 *Assuming the somewhere adaptive security of* ($\mathsf{SAdpGarbleCkt}, \mathsf{SAdpGarbleInp}$)*, we have* $\mathcal{H}_{4,(t',\alpha',\beta')} \overset{c}{\approx} \mathcal{H}_{4,(t,\alpha,\beta)}$ *for any two adjacent* (t',α',β') *and* (t,α,β)*.*

Proof Assume for the sake of contradiction that $\mathcal{H}_{4,(t',\alpha',\beta')}$ is distinguishable from $\mathcal{H}_{4,(t,\alpha,\beta)}$ with non-negligible probability. We use this to construct a reduction against the somewhere adaptive security of ($\mathsf{SAdpGarbleCkt}, \mathsf{SAdpGarbleInp}$).

Let $K' = \{(\bar{t},\overline{\alpha},\overline{\beta}) : (\bar{t},\overline{\alpha},\overline{\beta}) \preceq (t',\alpha',\beta')\}$. We define sub-hybrids and argue indistinguishability using the somewhere adaptive security.

- $\mathcal{H}'_1$: It is same as $\mathcal{H}_{4,(t',\alpha',\beta')}$ except for each $j \in [n] \setminus \{i\}$, the circuit computing the (t,α,β)-th entry in $\mathsf{C}_{\mathsf{inter}}\left[\left\{\mathsf{m}_k^{i\to j}\right\}_{k\in K'},\left\{\mathsf{m}_{k,0}^{i\to j},\mathsf{m}_{k,1}^{i\to j}\right\}_{k\in[4T]\setminus K',b\in\{0,1\}}\right]$ is simulated. Later, when s_j is obtained at the end of round-1, the output of the (t,α,β)-th circuit is computed and is used in obtaining $\widetilde{s}^{i\to j}$. $\mathcal{H}_{4,(t',\alpha',\beta')}$ is computationally indistinguishable to $\mathcal{H}'_1$ from the somewhere adaptive security.
- $\mathcal{H}'_2$: It is same as $\mathcal{H}'_2$ except that for each $j \in [n] \setminus \{i\}$, a string $\{\mathsf{m}_{(t,\alpha,\beta)}^{i\to j}\}$ is sampled randomly from $\{0,1\}^\lambda$ and is used as the output of (t,α,β)-th circuit instead of computing the output from s_j. In round-2, we set $\mathsf{m}_{(t,\alpha,\beta),\mathsf{PRG}_{\mathsf{mal}}(s_j)[(t,\alpha,\beta)]}^{i\to j} = \mathsf{m}_{(t,\alpha,\beta)}^{i\to j}$ and $\mathsf{m}_{(t,\alpha,\beta),1-\mathsf{PRG}_{\mathsf{mal}}(s_j)[k]}^{i\to j} \leftarrow \{0,1\}^\lambda$. This change is statistical and $\mathcal{H}'_1$ is distributed identically to $\mathcal{H}'_2$.
- $\mathcal{H}'_3$: This hybrid is same as $\mathcal{H}_{4,(t,\alpha,\beta)}$. $\mathcal{H}'_2$ is computationally indistinguishable to $\mathcal{H}'_3$ from the somewhere adaptive security of garbled circuits via a similar argument used to prove indistinguishability of $\mathcal{H}'_1$ and $\mathcal{H}_{4,(t',\alpha',\beta')}$.

This completes the proof of the claim. ■

- $\mathcal{H}_6$: In this hybrid, we change how the $\mathcal{F}_{\mathsf{zk}}$ functionality is emulated by the simulator in every call where $i \in H$ is acting as the verifier. In particular, for any $\mathsf{ssid} = (j \to i)$ where $i \in H$, the party P_i does not send $(\mu_0^{j\to i},\mu_1^{j\to i})$ to the ideal functionality and the simulator implicitly uses these inputs to compute the output of $\mathcal{F}_{\mathsf{zk}}$. This hybrid is identically distributed to $\mathcal{H}_5$.
- $\mathcal{H}_7$: In this hybrid, we change how the $\mathcal{F}_g$ functionality is emulated by the simulator in every call where $i \in H$ is acting as the receiver. In particular, for any $\mathsf{ssid} = i$, when $i \in H$ is

acting as the receiver in this execution, party P_i does not send $(s_i, \{\mathsf{rlab}_{k,b}^{j\to i}\})$ to the ideal functionality. Instead, the simulator implicitly uses these values in computing the output of $\mathcal{F}_g$. Again, this hybrid is identically distributed to $\mathcal{H}_6$.

- $\mathcal{H}_8$: In this hybrid, we make the following changes with respect to $\mathcal{H}_7$. For each $i \in H$,

  1. For each $j \in [n] \setminus \{\{i\} \cup H\}$, let $(\mathsf{ssid} = (j \to i), X^{j\to i}, W^{j\to i})$ be the message sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{zk}}$ functionality, and set $b^{j\to i} := R(X^{j\to i}, W^{j\to i})$. For all $j \in H \setminus \{i\}$, set $b^{j\to i} := 1$.
  2. For each $j \in [n] \setminus \{i\}$, sample
  $$\begin{aligned} \mu^{j\to i} &\leftarrow \{0,1\}^\lambda \\ \mathsf{rlab}_k^{j\to i} &\leftarrow \{0,1\}^\lambda \text{ for all } k \in [M] \end{aligned}$$
  3. Use $\mu^{j\to i}$ as the output of $\mathcal{F}_{\mathsf{zk}}$ execution for $\mathsf{ssid} = (j \to i)$ and $\{\mathsf{rlab}_k^{j\to i}\}$ as the output of the $\mathcal{F}_g$ execution for $\mathsf{ssid} = i$.
  4. Compute
  $$\widetilde{\mathsf{C}}^i_{\mathsf{wrap}} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\Big(1^\lambda, (\mathsf{C}^i_{\mathsf{wrap}}(\{b^{j\to i}\}_{j\in[n]\setminus\{i\}}, \{\widetilde{s}^{j\to i}\}_{j\in[n]\setminus\{i\}})), \Big\{\{\mu^{j\to i}\}_{j\in[n]\setminus\{i\}}, \{\mathsf{rlab}_k^{j}\}_{j\in[n]\setminus\{i\},k\in[M]}\Big\}\Big)$$
  where $\{\widetilde{s}^{j\to i}\}_{j\in[n]\setminus\{i\}}$ is computed by the simulator in emulating the $\mathcal{F}_g$ functionality.

  It follows directly from $|H|$ invocations of selective security of garbled circuits that $\mathcal{H}_7$ is computationally indistinguishable to $\mathcal{H}_8$.

- $\mathcal{H}_9$: In this hybrid, we make the following changes with respect to $\mathcal{H}_8$.

  For each $i \in H$, in this hybrid we complete an execution of Φ using honest parties' inputs and randomness. In this execution, the messages on behalf of the corrupted parties are generated via faithful execution. Specifically, we send $\{z_i\}_{i\in H}$ computed using honest parties' inputs and random coins, which starts the computation phase of Φ. In the computation phase, we generate honest parties' messages using the inputs and random coins of the honest parties and generate the messages of the each malicious party P_i by executing $\mathsf{Faithful}\left(i, \{z_j\}_{j\in[n]}, c_i\right)$. 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 $\gamma^* := \mathsf{st}^*[h]$.

  For each $i \in H$,

  1. If for some $i \in H$ and $j \in [n] \setminus \{i\}$, $b^{i\to j} = 0$ (i.e., the output of $\mathsf{C}^i_{\mathsf{wrap}}$ is $\perp$), then:
     (a) **for** each t from T down to 1,
        i. For each $k \in [N]$, sample $\mathsf{lab}_k^{i,t} \leftarrow \{0,1\}^\lambda$.
        ii. Compute
        $$\widetilde{\mathsf{P}}^{i,t} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, (\perp), \{\mathsf{lab}_k^{i,t}\}_{k\in[N]}\right).$$
     (b) Compute
     $$\widetilde{\mathsf{C}}^i_{\mathsf{wrap}} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, (\perp), \Big\{\{\mu^{j\to i}\}_{j\in[n]\setminus\{i\}}, \{\mathsf{rlab}_k^{j\to i}\}_{j\in[n]\setminus\{i\},k\in[M]}\Big\}\right).$$

    (c) Send $\left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \widetilde{\mathsf{C}}^i_{\mathsf{wrap}}\right)$ to every other party.

2. Otherwise:
    (a) For each $j \in [n] \setminus \{H \cup \{i\}\}$, recover $\mathsf{C_B}\left[\left\{\mathsf{m}^{j\to i}_{k,0}, \mathsf{m}^{j\to i}_{k,1}\right\}_{k\in[4T]}\right]$ from $W^{j\to i}$.
    (b) For each $k \in [N]$, set $\mathsf{lab}^{i,T+1}_k := 0^\lambda$.
    (c) For each t from T down to 1:
        i. Parse ϕ_t as (i^*, f, g, h).
        ii. Set $\alpha^* := \mathsf{st}^*[f]$, $\beta^* := \mathsf{st}^*[g]$, and $\gamma^* := \mathsf{st}^*[h]$.
        iii. For each $k \in [N]$, sample $\mathsf{lab}^{i,t}_k \leftarrow \{0,1\}^\lambda$.
        iv. If $i = i^*$ then compute

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

where $\omega_{t,\alpha^*,\beta^*} = \{\mathsf{m}^{j\to i}_{(t,\alpha^*,\beta^*),\mathsf{Z}[t]\oplus e_i[(t,\alpha^*,\beta^*)]}\}_{j\in[n]\setminus\{i\}}$.

        v. If $i \neq i^*$ then set $\delta_{\mathsf{Z}[t]} := \mathsf{m}^{i\to i^*}_{(t,\alpha^*,\beta^*)} \oplus \mathsf{lab}^{i,t+1}_h$ and $\delta_{1-\mathsf{Z}[t]} \leftarrow \{0,1\}^\lambda$. Set $ct := (\delta_0, \delta_1)$ and compute

$$\widetilde{\mathsf{P}}^{i,t} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}(1^\lambda, \left(ct, \{\mathsf{lab}^{i,t+1}_k\}_{k\in[N]\setminus\{h\}}\right), \{\mathsf{lab}^{i,t}_k\}_{k\in[N]}).$$

        vi. Compute

$$\widetilde{\mathsf{C}}^i_{\mathsf{wrap}} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, (\{\mathsf{lab}^{i,1}_k\}_{k\in[N]}), \left\{\{\mu^{j\to i}\}_{j\in[n]\setminus\{i\}}, \{\mathsf{rlab}^j_k\}_{j\in[n]\setminus\{i\},k\in[M]}\right\}\right)$$

Claim 6.2 *Assuming the selective security of garbling scheme for circuits, $\mathcal{H}_8 \overset{c}{\approx} \mathcal{H}_9$.*

Proof We show indistinguishability of the messages in $\mathcal{H}_8$ from $\mathcal{H}_9$ using a sequence of T sub-hybrids.

If for some $i \in H$ and $j \in [n] \setminus \{i\}$, $b^{i\to j} = 0$ (i.e., the output of $\mathsf{C}^i_{\mathsf{wrap}}$ is $\perp$), we can use a straightforward reduction to T invocations of the selective garbling scheme to show that $\mathcal{H}_8$ is computationally indistinguishable to distribution of messages in $\mathcal{H}_9$ for party P_i. Thus, we restrict our attention to those parties in H where this is not true.

We define for each $t \in [T]$ as hybrid $\mathcal{H}_{8,t}$ as follows:

- $\mathcal{H}_{8,t}$: Let $\phi_t = (i^*, f, g, h)$.
    - ∗ If $i^* \notin H$ then skip these changes. We make changes on how we generate the messages on behalf of party P_{i^*}. First, we sample for each $k \in [N]$, $\mathsf{lab}^{i^*,t}_k \leftarrow \{0,1\}^\lambda$. Second, we compute

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

where $\omega_{t,\alpha^*,\beta^*} = \{\mathsf{m}^{j\to i^*}_{(t,\alpha^*,\beta^*),\mathsf{Z}[t]\oplus e_{i^*}[(t,\alpha^*,\beta^*)]}\}_{j\in[n]\setminus\{i^*\}}$ where $\{\mathsf{lab}^{i^*,t+1}_{k,b}\}_{k\in[N],b\in\{0,1\}}$ are the honestly generates input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i^*,t+1}$ and $Z^t_{i^*}$ is updated state concatenated with masks at the end of the t-th round.

    * We make the following changes in how we generate messages for other honest parties P_i (i.e., $i \in H \setminus \{i^*\}$). First, for each $k \in [N]$, we sample $\mathsf{lab}_k^{i,t} \leftarrow \{0,1\}^\lambda$. Second, we set $\delta_{\mathsf{Z}[t]} := \mathsf{m}_{(t,\alpha^*,\beta^*)}^{i \to i^*} \oplus \mathsf{lab}_h^{i,t+1}$ and $\delta_{1-\mathsf{Z}[t]} \leftarrow \{0,1\}^\lambda$. Set $ct = (\delta_0, \delta_1)$. Third, we generate the garbled circuit

    $$\widetilde{\mathsf{P}}^{i,t} \leftarrow \mathsf{Sim}_{\mathsf{ckt}}(1^\lambda, \left(ct, \{\mathsf{lab}_{k,Z_i^t[k]}^{i,t+1}\}_{k \in [N] \setminus \{h\}}\right), \{\mathsf{lab}_k^{i,t}\}_{k \in [N]})$$

    where $\{\mathsf{lab}_{k,b}^{i,t+1}\}_{k \in [N]}$ are the honestly generated input labels for the garbled circuit $\widetilde{\mathsf{P}}^{i,t+1}$, and Z_i^t is updated state concatenated with masks at the end of the t-th round.

Note that the only difference between $\mathcal{H}_{8,t-1}$ and $\mathcal{H}_{8,t}$ is that in $\mathcal{H}_{8,t}$, we simulate $\widetilde{P}^{i,t}$ for every $i \in H$ whereas for in $\mathcal{H}_{8,t-1}$ we generate it honestly. To prove indistinguishability between $\mathcal{H}_{8,t-1}$ and $\mathcal{H}_{8,t}$ it is sufficient to show that the output we use in simulating $\widetilde{P}^{i,t}$ is same as the output of $\widetilde{P}^{i,t}$ in $\mathcal{H}_{8,t-1}$. We argue this separately for cases $i^* \in H$ and for all $i \in H \setminus \{i^*\}$.

- **Case-1:** $i^* \in H$: In this case, the only difference in the outputs are that, in $\mathcal{H}_{8,t}$, we set $\omega_{t,\alpha^*,\beta^*} = \{\mathsf{m}_{(t,\alpha^*,\beta^*),\mathsf{Z}[t] \oplus e_{i^*}[(t,\alpha^*,\beta^*)]}^{j \to i^*}\}_{j \in [n] \setminus \{i^*\}}$ where $\mathsf{m}_{(t,\alpha^*,\beta^*),\mathsf{Z}[t] \oplus e_{i^*}[(t,\alpha^*,\beta^*)]}^{j \to i^*}$ is extracted from $W^{j \to i^*}$ whereas in $\mathcal{H}_{8,t-1}$ this is obtained from the masks that are given as input. Note that these two values are indeed the same since $\mathcal{F}_{\mathsf{zk}}$ verifies that $\widetilde{\mathsf{C}}_{\mathsf{B}}^{j \to i}$ is computed correctly and Com is statistically binding commitment.
- **Case-2:** $i \in H \setminus \{i^*\}$**:** The fact that both the outputs are the same follows from inspection.

■

- $\mathcal{H}_{10}$: In this hybrid, we change how e_i is generated for every $i \in H$. Specifically, we sample $e_i \leftarrow \{0,1\}^{4T}$ instead of generating it as $c_i \oplus \mathsf{PRG}_{\mathsf{mal}}(s_i)$. Indistinguishability between $\mathcal{H}_9$ and $\mathcal{H}_{10}$ follows from $|H|$ invocations of the security of pseudorandom generator.

- $\mathcal{H}_{11}$: 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 parties' 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 $\mathsf{Faithful}(i, \{z_j\}_{j \in [n]}, c_i)$. (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 $\mathcal{H}_{10}$ and $\mathcal{H}_{11}$ follows directly from the malicious security of the protocol Φ.

  Note that $\mathcal{H}_{11}$ is distributed identically to the description of the simulator. This completes the proof of indistinguishability.

References

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

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Heidelberg, August 2013.

[ALSZ17] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. *Journal of Cryptology*, 30(3):805–858, July 2017.

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

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 479–488, 1996.

[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. Springer, Heidelberg, August 2016.

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

[BGJ$^+$17] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal mpc. Cryptology ePrint Archive, Report 2017/1088, 2017. `https://eprint.iacr.org/2017/1088`.

[BHP17] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 645–677. Springer, Heidelberg, November 2017.

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

[BL18] Fabrice Benhamouda and Huijia Lin. k-round mpc from k-round ot via garbled interactive circuits. To appear in Eurocrypt, 2018. `https://eprint.iacr.org/2017/1125`.

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

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

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

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

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

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

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

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

[COSV17] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Round-optimal secure two-party computation from trapdoor permutations. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 678–710. Springer, Heidelberg, November 2017.

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

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990.

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

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

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

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

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

[GMMM17] Sanjam Garg, Mohammad Mahmoody, Daniel Masny, and Izaak Meckler. On the round complexity of ot extension. Cryptology ePrint Archive, Report 2017/1187, 2017. `https://eprint.iacr.org/2017/1187`.

[GMPP16] Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 448–476. Springer, Heidelberg, May 2016.

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

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

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

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. To appear in Eurocrypt, 2018. `https://eprint.iacr.org/2017/1156`.

[HHPV17] Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. Cryptology ePrint Archive, Report 2017/1056, 2017. `http://eprint.iacr.org/2017/1056`.

[HIK07] Danny Harnik, Yuval Ishai, and Eyal Kushilevitz. How many oblivious transfers are needed for secure multiparty computation? In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 284–302. Springer, Heidelberg, August 2007.

[HIKN08] Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. OT-combiners via secure computation. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 393–411. Springer, Heidelberg, March 2008.

[HJO$^+$16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 149–178. Springer, Heidelberg, August 2016.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 132–150. Springer, Heidelberg, August 2011.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[JKK$^+$17] Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. Be adaptive, avoid overcommitting. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 133–163. Springer, Heidelberg, August 2017.

[JW16] Zahra Jafargholi and Daniel Wichs. Adaptive security of Yao's garbled circuits. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 433–458. Springer, Heidelberg, October / November 2016.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Heidelberg, August 2013.

[KRS16] Ranjit Kumaresan, Srinivasan Raghuraman, and Adam Sealfon. Network oblivious transfer. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 366–396. Springer, Heidelberg, August 2016.

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

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

[Nie07] Jesper Buus Nielsen. Extending oblivious transfers efficiently - how to get robustness almost for free. Cryptology ePrint Archive, Report 2007/215, 2007. `http://eprint.iacr.org/2007/215`.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, 2012.

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

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

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], 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}. [7]

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

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

A.4 The Common Reference/Random String Functionality

In the common reference string (CRS) model [CF01], 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 $\mathcal{F}^{D}_{CRS}$ that samples a string ρ from a pre-specified distribution D and sets ρ as the CRS. $\mathcal{F}^{D}_{CRS}$ is described in Figure 18.

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

$\mathcal{F}^{\mathbf{D}}_{\mathbf{CRS}}$ 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 18: The Common Reference String Functionality.

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

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

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 19. 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 $\mathcal{P} = \{P_1, \dots P_n\}$ (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 $(y_1, \dots y_n) \leftarrow f(x_1, \dots, x_n)$. 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 19: General Functionality.

B Construction of Somewhere Adaptive Garbling

We give the construction of somewhere adaptive garbled circuits in Figure 20. The construction makes use of a selective garbled circuit and a somewhere equivocal encryption scheme with block length set to $\max_i |\widetilde{C}_i|$, message length set to m and the equivocation parameter set to 1. We recall the definition of somewhere equivocal encryption from the work of [HJO+16] below.

Definition B.1 ([HJO+16]) *A somewhere equivocal encryption scheme with block-length s, message length n (in blocks) and equivocation parameter t (all polynomials in the security parameter) is a tuple of probabilistic polynomial algorithms* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SimEnc}, \mathsf{SimKey})$ *such that:*

- $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$: *It is a PPT algorithm that takes as input the security parameter (encoded in unary) and outputs a key* key.
- $\overline{c} \leftarrow \mathsf{Enc}(\mathsf{key}, m_1 \dots m_n)$: *It is a PPT algorithm that takes as input a key* key *and a vector of messages $\overline{m} = m_1 \dots m_n$ with each $m_i \in \{0,1\}^s$ and outputs a ciphertext $\overline{c}$.*
- $\overline{m} \leftarrow \mathsf{Dec}(\mathsf{key}, \overline{c})$: *It is a deterministic algorithm that takes as input a key* key *and a ciphertext $\overline{c}$ and outputs a vector of messages $\overline{m} = m_1 \dots m_n$.*
- $(\mathsf{state}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i \notin I}, I)$: *It is a PPT algorithm that takes as input a set of indices $I \subseteq [n]$ and a vector of messages $(m_i)_{i \notin I}$ and outputs a ciphertext $\overline{c}$ and a state* state.

- $\mathsf{key}' \leftarrow \mathsf{SimKey}(\mathsf{state}, (m_i)_{i \in I})$: *It is a PPT algorithm that takes as input the state information* state *and a vector of messages* $(m_i)_{i \in I}$ *and outputs a key* key'.

and satisfies the following properties:

Correctness. *For every* $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$, *for every* $\overline{m} \in \{0,1\}^{s \times n}$ *it holds that:*

$$\mathsf{Dec}(\mathsf{key}, \mathsf{Enc}(\mathsf{key}, \overline{m})) = \overline{m}.$$

Simulation with No Holes. *We require that the distribution of* $(\overline{c}, \mathsf{key})$ *computed via* $(\mathsf{state}, \overline{c}) \leftarrow \mathsf{SimEnc}(\overline{m}, \emptyset)$ *and* $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}, \emptyset)$ *to be identical to* $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and* $\overline{c} \leftarrow \mathsf{Enc}(\mathsf{key}, m_1 \ldots m_n)$. *In other words, simulation when there are no holes (i.e.,* $I = \emptyset$*) is identical to honest key generation and encryption.*

Security. *For any PPT adversary* $\mathcal{A}$, *there exists a negligible function* $\nu = \nu(\lambda)$ *such that:*

$$\left| \Pr[\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, 1) = 1] \right| \leq \nu(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}$ *is defined as follows:*

Experiment $\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}$

1. *The adversary* $\mathcal{A}$ *on input* 1^λ *outputs a set* $I \subseteq [n]$ *s.t.* $|I| < t$, *a vector* $(m_i)_{i \notin I}$, *and a challenge* $j \in [n] \setminus I$. *Let* $I' = I \cup \{j\}$.
2. - *If* $b = 0$, *compute* $\overline{c}$ *as follows:* $(\mathsf{state}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i \notin I}, I)$.
   - *If* $b = 1$, *compute* $\overline{c}$ *as follows:* $(\mathsf{state}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i \notin I'}, I')$.
3. *Send* $\overline{c}$ *to the adversary* $\mathcal{A}$.
4. *The adversary* $\mathcal{A}$ *outputs the set of remaining messages* $(m_i)_{i \in I}$.
   - *If* $b = 0$, *compute* key *as follows:* $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}, (m_i)_{i \in I})$.
   - *If* $b = 1$, *compute* key *as follows:* $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}, (m_i)_{i \in I'})$
5. *Send* key *to the adversary.*
6. $\mathcal{A}$ *outputs* b' *which is the output of the experiment.*

B.1 Security

Correctness and efficiency follows in a straightforward manner and we now argue the security. The simulated distribution is same as the distribution generated in $\mathcal{H}_3$ (defined later). We show that the real distribution is indistinguishable to the simulated distribution. We show this via a sequence of intermediate hybrids.

- $\mathcal{H}_1$: This corresponds to the real world distribution of $\widetilde{C}$ and $\widetilde{x}$.

```
$\mathsf{SAdpGarbleCkt}(1^\lambda, C)$: On input a circuit $C : \{0,1\}^n \to \{0,1\}^m$ do:
    1. Parse $C$ as $C_1, C_2, \ldots, C_m$.
    2. Sample $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$.
    3. For each $w \in [n]$ and $b \in \{0,1\}$ sample $\mathsf{lab}_{w,b} \leftarrow \{0,1\}^\lambda$. (We use $\overline{\mathsf{lab}}$ to denote
       $\{\mathsf{lab}_{w,b}\}_{w \in [n], b \in \{0,1\}}$.)
    4. for each $i \in [m]$ do:
        (a) Compute $\widetilde{C}_i \leftarrow \mathsf{Garble}\left(1^\lambda, C_i, \overline{\mathsf{lab}}\right)$.
    5. Compute $\overline{c} \leftarrow \mathsf{Enc}(\mathsf{key}, \{\widetilde{C}_i\}_{i \in [m]})$.
    6. Output $\widetilde{C} := \overline{c}$ and $\mathsf{state} := (\mathsf{key}, \overline{\mathsf{lab}})$.
$\mathsf{SAdpGarbleInp}(\mathsf{state}, x)$ : On input the state $\mathsf{state}$ and a string $x \in \{0,1\}^n$ do:
    1. Parse $\mathsf{state}$ as $(\mathsf{key}, \{\mathsf{lab}_{w,b}\}_{w \in [n], b \in \{0,1\}})$
    2. Output $\widetilde{x} := \left(\mathsf{key}, \{\mathsf{lab}_{w,x[w]}\}_{w \in [n]}\right)$.
$\mathsf{SAdpEvalCkt}(\widetilde{C}, \widetilde{x})$ : On input garbled circuit $\widetilde{C}$, and garbled input $\widetilde{x}$ do:
    1. Parse $\widetilde{C}$ as $\overline{c}$ and $\widetilde{x}$ as $\left(\mathsf{key}, \{\mathsf{lab}_w\}_{w \in [n]}\right)$.
    2. Compute $\{\widetilde{C}_i\}_{i \in [m]} := \mathsf{Dec}(\mathsf{key}, \overline{c})$.
    3. for each $i \in [m]$ do:
        (a) Compute $y_i := \mathsf{Eval}(\widetilde{C}_i, \{\mathsf{lab}_w\}_{w \in [n]})$.
    4. Output $y_1 \ldots y_m$.
```

Figure 20: Distribution-Indistinguishable Adaptive Garbled Circuits

- $\mathcal{H}_2$: In this hybrid, we generate the garbled circuit $\widetilde{C}$ and the garbled input $\widetilde{x}$ as follows. **Generating $\widetilde{C}$:**

```
1. Let $C$ be $C_1, \ldots, C_j, C_{j+1}, \ldots, C_m$.
2. For each $w \in [n]$ and $b \in \{0,1\}$ sample $\mathsf{lab}_{w,b} \leftarrow \{0,1\}^\lambda$. (We use $\overline{\mathsf{lab}}$ to denote
   $\{\mathsf{lab}\}_{w \in [n], b \in \{0,1\}}$.)
3. for each $\underline{i \in [m] \setminus \{j\}}$ do:
    (a) Compute $\widetilde{C}_i \leftarrow \mathsf{Garble}\left(1^\lambda, C_i, \overline{\mathsf{lab}}\right)$.
4. Compute $\underline{(\overline{c}, \mathsf{state}_{\mathsf{Sim}}) \leftarrow \mathsf{SimEnc}(\{\widetilde{C}_i\}_{i \neq [j]}, \{j\})}$.
5. Output $\widetilde{C} := \overline{c}$ and $\mathsf{state} := (\mathsf{state}_{\mathsf{Sim}}, \overline{\mathsf{lab}})$.
```

Generating $\widetilde{x}$:

```
1. Parse $\mathsf{state}$ as $(\mathsf{state}_{\mathsf{Sim}}, \{\mathsf{lab}_{w,b}\}_{w \in [n], b \in \{0,1\}})$.
2. Compute $\widetilde{C}_j \leftarrow \mathsf{Garble}\left(1^\lambda, C_j, \overline{\mathsf{lab}}\right)$.
```

3. Set $\underline{\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}_{\mathsf{Sim}}, \widetilde{C}_j)}$.
4. Output $\mathsf{key}, \{\mathsf{lab}_{w,x[w]}\}_{w \in [n]}$.

It follows directly from the simulation with no holes and the security properties of somewhere equivocal encryption that $\mathcal{H}_1$ is computationally indistinguishable to $\mathcal{H}_2$.

- $\mathcal{H}_3$: In this hybrid, we generate the garbled circuit $\widetilde{C}$ as in $\mathcal{H}_2$ but generate the garbled input $\widetilde{x}$ as follows.
  Generating $\widetilde{x}$:

  1. Parse state as $(\mathsf{state}_{\mathsf{Sim}}, \{\mathsf{lab}_{w,b}\}_{w \in [n], b \in \{0,1\}})$.
  2. Compute $\underline{\widetilde{C}_j \leftarrow \mathsf{Sim}_{\mathsf{ckt}}\left(1^\lambda, 1^{|C_j|}, C_j(x), \{\mathsf{lab}_{w,x[w]}\}_{w \in \mathsf{inp}(C)}\right)}$.
  3. Set $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}_{\mathsf{Sim}}, \widetilde{C}_{j+1})$.
  4. Output $\mathsf{key}, \{\mathsf{lab}_{w,x[w]}\}_{w \in [n]}$.

  It follows from the selective security of garbled circuits that $\mathcal{H}_2$ is computationally indistinguishable to $\mathcal{H}_3$. $\mathcal{H}_3$ represents the simulated distribution.