Reusable Two-Round MPC from DDH

Saikrishna Badrinarayanan* James Bartusek† Sanjam Garg‡ Daniel Masny§
Pratyay Mukherjee¶

February 13, 2020

Abstract

We present a reusable two-round multi-party computation (MPC) protocol from the Decisional Diffie Hellman assumption (DDH). In particular, we show how to upgrade *any* secure two-round MPC protocol to allow reusability of its first message across multiple computations, using Homomorphic Secret Sharing (HSS) and pseudorandom functions in NC^1— each of which can be instantiated from DDH.

In our construction, if the underlying two-round MPC protocol is secure against semi-honest adversaries (in the plain model) then so is our reusable two-round MPC protocol. Similarly, if the underlying two-round MPC protocol is secure against malicious adversaries (in the common random/reference string model) then so is our reusable two-round MPC protocol. Previously, such reusable two-round MPC protocols were only known under assumptions on lattices.

At a technical level, we show how to upgrade any two-round MPC protocol to a *first message succinct* two-round MPC protocol, where the first message of the protocol is generated *independently* of the computed circuit (though it is not reusable). This step uses homomorphic secret sharing (HSS) and low-depth pseudorandom functions. Next, we show a generic transformation that upgrades any first message succinct two-round MPC to allow for reusability of its first message.

1 Introduction

Motivating Scenario. Consider the following setting: a set of n hospitals publish encryptions of their sensitive patient information $x_1, \ldots, x_n$. At a later stage, for the purposes of medical research, they wish to securely evaluate a circuit C_1 on their joint data by publishing just one additional message - that is, they wish to jointly compute $\mathsf{C}_1(x_1, \ldots, x_n)$ by each broadcasting a single message, without revealing anything more than the output of the computation. Can they do so? Furthermore, what if they want to additionally compute circuits $\mathsf{C}_2, \mathsf{C}_3 \ldots$ at a later point on the same set of inputs?

Seminal results on secure multi-party computation (MPC) left quite a bit to be desired when considering the above potential application. In particular, the initial construction of secure multi-party computation by Goldreich, Micali and Wigderson [GMW87] required parties to interact over a large number of rounds. Even though the round complexity was soon reduced to a constant by Beaver, Micali and Rogaway [BMR90], these protocols fall short of achieving the above vision, where interaction is reduced to the absolute minimum.

Making progress towards this goal, Garg et al. [GGHR14] gave the first constructions of *two-round* MPC protocols, assuming indistinguishability obfuscation [GGH+13] (or, witness encryption [GLS15, GGSW13])

*Visa Research, bsaikrishna7393@gmail.com

†University of California, Berkeley, bartusek.james@gmail.com

‡University of California, Berkeley, sanjamg@berkeley.edu. Research supported in part from DARPA/ARL SAFEWARE Award W911NF15C0210, AFOSR Award FA9550-15-1-0274, AFOSR YIP Award, DARPA and SPAWAR under contract N66001-15-C-4065, a Hellman Award 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

§Visa Research, Daniel.Masny@rub.de

¶Visa Research, pratyay85@protonmail.com

and one-way functions.[1] A very nice feature of the Garg et al. construction is that the first round message is indeed reusable across multiple executions, thereby achieving the above vision. Follow up works realized two-round MPC protocols based on significantly weaker computational assumptions. In particular, two-round MPC protocols based on LWE were obtained [MW16, BP16, PS16], followed by a protocol based on bilinear maps [GS17, BF01, Jou04]. Finally, this line of work culminated with the recent works of Benmahouda and Lin [BL18] and Garg and Srinivasan [GS18], who gave constructions based on the minimal assumption that two-round oblivious transfer (OT) exists.

However, in these efforts targeting two-round MPC protocols with security based on weaker computational assumptions, compromises were made in terms of reusability. In particular, among the above mentioned results only the obfuscation based protocol of Garg et al. [GGHR14] and the lattice based protocols [MW16, BP16, PS16] offer reusability of the first message across multiple executions. Reusability of the first round message is quite desirable. In fact, even in the two-party setting, this problem has received significant attention and has been studied under the notion of non-interactive secure computation [IKO$^+$11, AMPR14, MR17, BJOV18, CDI$^+$19]. In this setting, a receiver first publishes an encryption of its input and later, any sender may send a single message (based on an arbitrary circuit) allowing the receiver to learn the output of the circuit on its input. The multiparty case, which we study in this work, can be seen as a natural generalization of the problem of non-interactive secure computation. In this work we ask:

Can we obtain reusable *two-round MPC protocols from assumptions not based on lattices?*

1.1 Our Result

In this work, we answer the above question by presenting a general compiler that obtains reusable two-round MPC, starting from any two-round MPC and using homomorphic secret sharing (HSS) [BGI16] and pseudorandom functions in NC^1. In a bit more detail, our main theorem is:

Theorem 1.1 (Main Theorem)**.** *Let $\mathcal{X} \in \{$semi-honest in plain model, malicious in common random/reference sting model$\}$. Assuming the existence of a two-round $\mathcal{X}$-MPC protocol, an HSS scheme, and pseudorandom functions in NC^1, there exists a* reusable *two-round $\mathcal{X}$-MPC protocol.*

We consider the setting where an adversary can corrupt an arbitrary number of parties. We assume that parties have access to a broadcast channel.

Benmahouda and Lin [BL18] and Garg and Srinivasan [GS18] showed how to build a two-round MPC protocol from the DDH assumption. The works of Boyle et al. [BGI16, BGI17] constructed an HSS scheme assuming DDH. Instantiating the primitives in the above theorem, we get the following corollary:

Corollary 1.2. *Let $\mathcal{X} \in \{$semi-honest in plain model, malicious in common random/reference sting model$\}$. Assuming DDH, there exists a* reusable *two-round $\mathcal{X}$-MPC protocol.*

Previously, constructions of reusable two-round MPC were only known assuming indistinguishability obfuscation [GGHR14, GS17] (or, witness encryption [GLS15, GGSW13]) or were based on multi-key fully-homomorphic encryption (FHE) [MW16, PS16, BP16]. Furthermore, one limitation of the FHE-based protocols is that they are in the CRS model even for the setting of semi-honest adversaries.

We note that the two-round MPC protocols cited above additionally achieve overall communication independent of the computed circuit. This is not the focus of this work. Instead, the aim of this work is to realize two-round MPC with reusability, without relying on lattices. As per our current understanding, MPC protocols with communication independent of the computed circuit are only known using lattice techniques (i.e., FHE [Gen09]). Interestingly, we use HSS, which was originally developed to improve communication efficiency in two-party secure computation protocols, to obtain reusability.

[1] The Garg et al. paper required other assumptions. However, since then they have all been shown to be implied by indistinguishability obfuscation and one-way functions.

First Message Succinct Two-Round MPC. At the heart of this work is a construction of a *first message succinct* (FMS) two-round MPC protocol— that is, a two-round MPC protocol where the first message of the protocol is computed independently of the circuit being evaluated. In particular, the parties do not need to know the description of the circuit that will eventually be computed over their inputs in the second round. Furthermore, parties do not even need to know the *size* of the circuit to be computed in the second round.[2] This allows parties to publish their first round messages and later compute any arbitrary circuit on their inputs. Formally, we show the following:

Theorem 1.3. *Let $\mathcal{X} \in \{$semi-honest in plain model, malicious in common random/reference sting model$\}$. Assuming DDH, there exists a first message succinct two-round $\mathcal{X}$-MPC protocol.*

Such protocols were previously only known based on iO [GGHR14, DHRW16] or assumptions currently needed to realize FHE [MW16, BP16, PS16, ABJ+19]. Note that for the learning-with-errors (LWE) based versions of these protocols, the first message can only be computed knowing the depth (or, an upper bound on the maximum depth) of the circuit to be computed. We find the notion of first message succinct two-round MPC quite natural and expect it be relevant for several other applications. In addition to using HSS in a novel manner, our construction benefits from the powerful garbling techniques realized in recent works [LO13, GHL+14, GLOS15, GLO15, CDG+17, DG17b].

From First Message Succinctness to Reusability. On first thought, the notion of first message succinctness might seem like a minor enhancement. However, we show that this "minor looking" enhancement is sufficient to enable reusable two-round MPC (supporting arbitrary number of computations) generically. More formally:

Theorem 1.4. *Let $\mathcal{X} \in \{$semi-honest in plain model, malicious in common random/reference sting model$\}$. Assuming a first message succinct two-round MPC protocol, there exists a* reusable *two-round $\mathcal{X}$-MPC protocol.*

2 Technical Overview

In this section, we highlight our main ideas for obtaining reusability in two-round MPC. Our construction is achieved in three steps. Our starting point is the recently developed primitive of Homomorphic Secret Sharing (HSS), which realizes the following scenario. A secret s is shared among two parties, who can then non-interactively evaluate a function f over their respective shares, and finally combine the results to learn $f(s)$, but nothing more.

2.1 Step 1: Overview of the scHSS Construction

First, we show how to use a "standard" HSS (for only two parties, and where the reconstruction algorithm is simply addition) to obtain a new kind of HSS, which we call *sharing compact HSS* (scHSS). The main property we achieve with scHSS is the ability to share a secret among n parties, for any n, while maintainting compactness of the share size. In particular, as in standard HSS, the sharing algorithm will be independent of the circuit that will be computed on the shares. We actually obtain a few other advantages over constructions of standard HSS [BGI16, BGI17], namely, we get negligible rather than inverse polynomial evaluation error, and we can support computations of any polynomial-size circuit. To achieve this, we sacrifice compactness of the *evaluated* shares, simplicity of the reconstruction algorithm, and security for multiple evaluations. However, it will only be crucial for us that multiple parties can participate, and that the *sharing* algorithm is compact.

[2] Note that this requirement is more stringent than just requiring that the size of the first round message is independent of the computed circuit, which can be achieved using laconic OT [CDG+17] for any two-round MPC protocol.

The approach: A sharing-compact HSS scheme consists of three algorithms, $\mathsf{Share}, \mathsf{Eval}$, and Dec. Our construction follows the compiler of [GS18] that takes an arbitrary MPC protocol and squishes it to two rounds. At a high level, to share a secret x among n parties, we have the Share algorithm first compute an n-party additive secret sharing $x_1, \ldots, x_n$ of x. Then, it runs the first round of the squished n-party protocol on behalf of each party j with input x_j.[3] Finally, it sets the j'th share to be all of the first round messages, plus the secret state of the j'th party. The Eval algorithm run by party j will simply run the second round of the MPC, and output the resulting message. The Dec algorithm takes all second round messages and reconstructs the output.

Recall that we aim for a sharing-compact HSS, which in particular means that the Share algorithm must be independent of the computation supported during the Eval phase. Thus, the first observation that makes the above approach viable is that the first round of the two-round protocol that results from the [GS18] compiler is *independent* of the particular circuit being computed. Unfortunately, it is *not* generated independently of the *size* of the circuit to be computed, so we must introduce new ideas to remove this size dependence.

The [GS18] compiler: Before further discussing the size dependence issue, we recall the [GS18] compiler. The compiler is applied to any *conforming* MPC protocol, a notion defined in [GS18].[4] Roughly, a conforming protocol operates via a sequence of actions $\phi_1, \ldots, \phi_T$. At the beginning of the protocol, each party j broadcasts a one-time pad of their input, and additionally generates some secret state v_j. The encrypted inputs are arranged into a global public state st, which will be updated throughout the protocol. At each step t, the action $\phi_t = (j, f, g, h)$ is carried out by having party j broadcast the bit $\gamma_t := \mathsf{NAND}(\mathsf{st}_f \oplus v_{j,f}, \mathsf{st}_g \oplus v_{j,g}) \oplus v_{j,h}$. Everybody then updates the global state by setting $\mathsf{st}_h := \gamma_t$. We require that the transcript of the protocol is publicly decodable, so that after the T actions are performed, anybody can learn the (shared) output by inspecting st.

Now, the [GS18] compiler works as follows. In the first round of the compiled protocol, each party runs the first round of the conforming protocol and broadcasts a one-time pad of their input. In the second round, each party generates a set of garbled circuits that non-interactively implement the computation phase of the conforming protocol. In particular, this means that an evaluator can use the garbled circuits output by each party to carry out each action $\phi_1, \ldots, \phi_T$, learn the resulting final st, and recover the output. The garbled circuits operate as follows. Each garbled circuit for party j takes as input the public state st, and outputs information that allows recovery of input labels for party j's next garbled circuit, corresponding to an updated version of the public state. To facilitate this, the initial private state of each party must be hard-coded into each of their garbled circuits.

In more detail, consider a particular round t and action $\phi_t = (j^*, f, g, h)$. Each party will output a garbled circuit for this round. We refer to party j^* as the "speaking" party for this round. Party j^*'s garbled circuit will simply use its private state to compute the appropriate NAND gate and update the public state accordingly, outputting the correct labels for party j^*'s next garbled circuit, and the bit γ_t to be broadcast. It remains to show how the garbled circuit of each party $j \neq j^*$ can incorporate this bit γ_t, revealing the correct input label for *their* next garbled circuit. We refer to party j as the "listening" party. In [GS18], this was facilitated by the use of a two-round oblivious transfer (OT). In the first round, each pair of parties (j, j^*) engages in the first round of multiple OT protocols with j acting as the sender and j^* acting as the receiver. Specifically, j^* sends a set of receiver messages to party j. Then during action t, party j's garbled circuit responds with j's sender message, where the sender's two strings are garbled input labels $\mathsf{lab}_0, \mathsf{lab}_1$ of party j's next garbled circuit. Party j^*'s garbled circuit reveals the randomness used to produce the receiver's message with the appropriate receiver bit γ_t. This allows for public recovery of the label lab_{γ_t}.

However, note that each of the T actions requires its own set of OTs to be generated in the first round. Each is then "used up" in the second round, as the receiver's randomness is revealed in the clear. This is precisely what makes the first round of the resulting MPC protocol depend on the *size* of the circuit to be computed: the parties must engage in the first round of $\Omega(T)$ oblivious transfers during the first round of

[3] Actually, we use an $n\lambda$-party MPC protocol, for reasons that will become clear later in this overview.

[4] We tweak the notion slightly here, so readers familiar with [GS18] may notice some differences in this overview.

the MPC protocol.

Pair-wise correlations: As observed also in [GIS18], the point of the first round OT messages was to set up pair-wise correlations between parties that were then exploited in the second round to facilitate the transfer of a bit from party j^*'s garbled circuit to party j's garbled circuit. For simplicity, assume for now that when generating the first round, the parties j and j^* already know the bit γ_t that is to be communicated during action t. This is clearly not the case, but this issue is addressed in [GS18, GIS18] (and here) by generating four sets of correlations, corresponding to each of the four possible settings of the two bits of the public state (α, β) at the indices (f, g) corresponding to action $\phi_t = (j^*, f, g, h)$.

Now observe that the following correlated randomness suffices for this task. Party j receives uniformly random strings $z^{(0)}, z^{(1)} \in \{0,1\}^\lambda$, and party j^* receives the string $z^{(*)} := z^{(\gamma_t)}$. Recall that party j has in mind garbled input labels $\mathsf{lab}_0, \mathsf{lab}_1$ for its next garbled circuit, and wants to reveal lab_{γ_t} in the clear, while keeping $\mathsf{lab}_{1-\gamma_t}$ hidden. Thus, party j's garbled circuit will simply output $(\mathsf{lab}_0 \oplus z^{(0)}, \mathsf{lab}_1 \oplus z^{(1)})$, and party j^*'s garbled circuit outputs $z^{(*)}$. Now, instead of generating first round OT messages, the Share algorithm could simply generate all of the pair-wise correlations and include them as part of the shares. Of course, the number of correlations necessary still depends on T, so we will need the Share algorithm to produce *compact* representations of these correlations.

Compressing using constrained PRFs: Consider a pair of parties (j, j^*), and let T_{j^*} be the set of actions where j^* is the speaking party. We need the output of Share to (implicitly) include random strings $\{z_t^{(0)}, z_t^{(1)}\}_{t \in T_{j^*}}$ in j's share and $\{z_t^{(\gamma_t)}\}_{t \in T_{j^*}}$ in j^*'s share. The first set of strings would be easy to represent compactly with a PRF key k_j, letting $z_t^{(b)} := \mathsf{PRF}(k_j, (t, b))$. However, giving the key k_j to party j^* would reveal too much, as it is imperative that we keep $\{z_t^{(1-\gamma_t)}\}_{t \in T_{j^*}}$ hidden from party j^*'s view. We could instead give party j^* a *constrained* version of the key k_j that only allows j^* to evaluate $\mathsf{PRF}(k_j, \cdot)$ on points (t, γ_t). We expect that this idea can be made to work, and one could hope to present a construction based on the security of (single-key) constrained PRFs for constraints in NC^1 (plus a standard PRF computable in NC^1). Such a primitive was achieved in [AMN+18] based on assumptions in a traditional group, however, we aim for a construction from weaker assumptions.

Utilizing HSS: Inspired by [BCGI18, BCG+19], we take a different approach based on HSS. Consider sharing the PRF key k_j between parties j and j^*, producing shares sh_j and sh_{j^*}, and additionally giving party j the key k_j in the clear. During action t, we have parties j and j^* (rather, their garbled circuits) evaluate the following function on their respective shares: if $\gamma_t = 0$, output 0^λ and otherwise, output $\mathrm{PRF}(k_j, t)$. Assuming that the HSS evaluation is correct, and using the fact that HSS reconstruction is *additive* (over $\mathbb{Z}_2$), this produces a pair of outputs (y_j, y_{j^*}) such that if $\gamma_t = 0$, $y_j \oplus y_{j^*} = 0^\lambda$, and if $\gamma_t = 1$, $y_j \oplus y_{j^*} = \mathsf{PRF}(k_j, t)$. Now party j sets $z_t^{(0)} := y_j$ and $z_t^{(1)} := y_j \oplus \mathsf{PRF}(k_j, t)$, and party j^* sets $z_t^{(*)} := y_{j^*}$. This guarantees that $z_t^{(*)} = z_t^{(\gamma_t)}$ and that $z_t^{(1-\gamma_t)} = z_t^{(*)} \oplus \mathsf{PRF}(k_j, t)$, which should be indistinguishable from random to party j^*, who doesn't have k_j in the clear.

Tying loose ends: This approach works, except that, as alluded to before, party j's garbled circuit will not necessarily know the bit γ_t when evaluating its HSS share. This is handled by deriving γ_t based on public information (some bits α, β of the public shared state), and the private state of party j^*. Since party j^*'s private state cannot be public information, this derivation must happen within the HSS evaluation, and in particular, the secret randomness that generates j^*'s private state must be part of the secret shared via HSS. In our construction, we compile a conforming protocol where each party j^*'s randomness can be generated by a PRF with key s_{j^*}. Thus, we can share the keys (k_j, s_{j^*}) between parties j and j^*, allowing them to compute output shares with respect to the correct γ_t. Finally, note that the computation performed by HSS essentially only consists of PRF evaluations. Thus, assuming a PRF in NC1 (which follows from DDH [NR97]), we only need to make use of HSS that supports evaluating circuits in NC1, which also follows from DDH [BGI16, BGI17].

Dealing with the $1 - 1/\mathsf{poly}$ correctness of HSS: We are not quite done, since the [BGI16, BGI17] constructions of HSS only achieve correctness with $1 - 1/\mathsf{poly}$ probability. At first glance, this appears to be straightforward to fix. To complete action $\phi_t = (j^*, f, g, h)$, simply repeat the above λ times, now generating sets $\{z^{(0)}_{t,p}, z^{(1)}_{t,p}\}_{p\in[\lambda]}$ and $\{z^{(*)}_{t,p}\}_{p\in[\lambda]}$, using the values $\{\mathsf{PRF}(k_j, (t,p))\}_{p\in[\lambda]}$. Party j now masks the same labels $\mathsf{lab}_0, \mathsf{lab}_1$ with λ different masks, and to recover lab_{γ_t}, one can unmask each value and take the most frequently occurring string to be the correct label. This does ensure that our scHSS scheme is correct except with negligible probability.

Unfortunately, the $1/\mathsf{poly}$ correctness actually translates to a *security* issue with the resulting scHSS scheme. In particular, it implies that an honest party's evaluated share is indistiguishable from a simulated evaluated share with probability only $1-1/\mathsf{poly}$. To remedy this, we actually use an $n\lambda$-party MPC protocol, and refer to each of the $n\lambda$ parties as a "virtual" party. The Share algorithm now additively secret shares the secret x into $n\lambda$ parts, and each of the n real parties participating in the scHSS receives the share of λ virtual parties. We are then able to show that for any set of honest parties, with overwhelming probability, there will exist at least one corresponding virtual party that is "simulatable". The existence of a single simulatable virtual party is enough to prove the security of our construction.

At this point it is important to point out that, while the above strategy suffices to prove our construction secure for a single evaluation (where the circuit evaluated can be of any arbitrary polynomial size), it does *not* imply that our construction achieves reusability, in the sense that the shares output by Share may be used to evaluate any unbounded polynomial number of circuits. Despite the fact that the PRF keys shared via HSS should enable the parties to generate an unbounded polynomial number of pair-wise correlations, the $1/\mathsf{poly}$ evaluation error of the HSS will eventually break simulation security. Fortunately, as alluded to before, the property of sharing-compactness actually turns out to be enough to bootstrap our scheme into a truly reusable MPC protocol. The key ideas that allow for this will be discussed in Section 2.3.

2.2 Step 2: From scHSS to FMS MPC

In the second step, we use a scHSS scheme to construct a first message succinct two-round MPC protocol (in the rest of this overview we will call it FMS MPC). The main feature of a scHSS scheme is that its Share algorithm is independent of the computation that will be performed on the shares. Intuitively, this is very similar to the main feature offered by a FMS MPC protocol, in that the first round is independent of the circuit to be computed. Now, suppose that we have an imaginary trusted entity that learns everyone's input $(x_1, \ldots, x_n)$ and then gives each party i a share sh_i computed as $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(x_1 \| \ldots \| x_n)$. Note that, due to sharing-compactness this step is independent of the circuit C to be computed by the FMS MPC protocol. After receiving their shares, each party i runs the scHSS evaluation circuit $\mathsf{Eval}(i, \mathsf{C}, \mathsf{sh}_i)$ to obtain their own output share y_i, and then broadcasts y_i . Finally, on receiving all the output shares $(y_1, \ldots, y_n)$, everyone computes $y := \mathsf{C}(x_1, \ldots x_n)$ by running the decoding procedure of scHSS: $y := \mathsf{Dec}(y_1, \ldots, y_n)$.

A straightforward *three*-round protocol. Unfortunately, we do *not* have such a trusted entity available in the setting of FMS MPC. A natural approach to resolve this would be to use any standard two-round MPC protocol (from now on we refer to such a protocol as vanilla MPC) to realize the Share functionality in a distributed manner. However, since the vanilla MPC protocol would require at least two rounds to complete, this straightforward approach would incur one additional round. This is inevitable, because the parties receive their shares only at the end of the second round. Therefore, an additional round of communication (for broadcasting the output shares y_i) would be required to complete the final protocol.

Garbled circuits to the rescue. Using garbled circuits, we are able to squish the above protocol to operate in only *two* rounds. The main idea is to have each party i additionally send a garbled circuit $\widetilde{\mathsf{C}}_i$ in the second round. Each $\widetilde{\mathsf{C}}_i$ garbles a circuit that implements $\mathsf{Eval}(i, \mathsf{C}, \cdot)$. Given the labels for sh_i, $\widetilde{\mathsf{C}}_i$ can be evaluated to output $y_i \leftarrow \mathsf{Eval}(i, \mathsf{C}, \mathsf{sh}_i)$. Note that, if it is ensured that every party receives all the garbled circuits *and* all the correct labels after the second round, they can obtain all $(y_1, \ldots, y_n)$, and compute the

final output y without further communication. The only question left now is how the correct labels are communicated within two rounds.

Tweaking vanilla MPC to output labels. For communicating the correct labels, we slightly tweak the functionality computed by the vanilla MPC protocol. In particular, instead of using it just to compute the shares $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$, we have the vanilla MPC protocol compute a slightly different functionality that first computes the shares, and rather than outputting them directly, outputs the corresponding correct labels for everyone's shares.[5] This is enabled by having each party provide a random value r_i, which is used to generate the labels, as an additional input to D. Therefore, everyone's correct labels are now available after the completion of the second round of the vanilla MPC protocol. Recall that parties also broadcast their garbled circuits along with the second round of the vanilla MPC. Each party i, on receiving all $\widetilde{\mathsf{C}}_1, \ldots \widetilde{\mathsf{C}}_n$ and all correct labels, evaluates to obtain $(y_1, \ldots, y_n)$ and then computes the final output y.[6]

2.3 Step 3: From FMS MPC to Reusable MPC

Finally, in this third step, we show how FMS MPC can be used to construct *reusable* two-round MPC, where the first message of the protocol can be reused across multiple computations.

We start with the observation that a two-round FMS MPC protocol allows us to compute arbitrary sized circuits after completion of the first round. This offers a limited form of (bounded) reusability, in that all the circuits to be computed could be computed together as a single circuit. However, once the second round is completed, no further computation is possible. Thus, the main challenge is how to leverage the ability to compute a single circuit of unbounded size to achieve *unbounded* reusability. Inspired by ideas from [DG17b], we address this challenge by using the ideas explained in Step 2 (above) repeatedly. For the purposes of this overview, we first explain a simpler version of our final protocol, in which the second round is expanded into multiple rounds. A key property of this protocol is that, using garbled circuits, those expanded rounds can be squished back into just one round (just like we did in Step 2) while preserving reusability.

Towards reusability: a multi-round protocol. The fact that FMS MPC does not already achieve reusability can be re-stated as follows: the first round of FMS MPC (computed using an algorithm MPC_1) can only be used for a single second round execution (using an algorithm MPC_2). To resolve this issue, we build a GGM-like [GGM84] tree-based mechanism that generates a *fresh* FMS first round message for each circuit to be computed, while ensuring that no FMS first round message is reused.

The first round of our final two-round reusable protocol, as well the multi-round simplified version, simply consists of the first round message corresponding to the *root* level (of the GGM tree) instance of the FMS protocol. We now describe the subsequent rounds (to be squished to a single second round later) of our multi-round protocol.

Intuitively, parties iteratively use an FMS instance at a particular level of the binary tree (starting from the root) to generate two new first-round FMS messages corresponding to the next level of the tree. The leaf FMS protocol instances will be used to compute the actual circuits. The root to leaf path traversed to compute a circuit C is decided based on the description of the circuit C itself.[7]

In more detail, parties first send the second round message of the root (0'th) level FMS protocol instance for a fixed circuit N (independent of the circuit C to be computed) that samples and outputs "left" and "right" MPC_1 messages using the same inputs that were used in the root level FMS. Now, depending on the first bit of the circuit description, parties choose either the left (if the first bit is 0) or the right (if the

[5] It is important to note that the set of garbled labels corresponding to some input x hides the actual string x. Hence, outputting all the labels instead of specific shares enables everyone to obtain the desired output without any further communication, but also does not compromise security.

[6] We remark that, in the actual protocol each party i sends their labels, encrypted, along with the garbled circuit $\widetilde{\mathsf{C}}_i$ in the second round. The vanilla MPC protocol outputs the correct sets of decryption keys based on the shares, which allows everyone to obtain the correct sets of labels, while the other labels remain hidden.

[7] We actually use the string whose first λ bits are the *size* of C, and the remaining bits are the description of C. This is to account for the possibility that one circuit may be a prefix of another.

first bit is 1) MPC_1 messages for the next (1st) level. Now using the chosen FMS messages, parties generate the MPC_2 message for the same circuit N as above. This results in two more fresh instances of the MPC_1 messages for the next (2nd) level. As mentioned before, this procedure is continued until the leaf node is reached. At that point the MPC_2 messages are generated for the circuit C that the parties are interested in computing.

Note that, during the evaluation of two different circuits (each associated with a different leaf node), a certain number of FMS protocol instances might get *re-executed*. However, our construction ensures that this is merely a re-execution of a fixed circuit with the exact same input/output behavior each time. This guarantees that no FMS message is reused (even though it might be re-executed). Finally, observe that this process of iteratively computing more and more MPC_1 messages for the FMS protocol is only possible because the generation of the first message of an FMS protocol can be performed independently of the circuit that gets computed in the second round. In particular, the circuit N computes two more MPC_1 messages on behalf of each party.

Squishing the multiple rounds: using ideas in Step 2 iteratively. We take an approach similar to Step 2, but now starting with a two-round FMS MPC (instead of a vanilla MPC). In the second round, each party will send a sequence of garbled circuits where each garbled circuit will complete one instance of an FMS MPC which generates labels for the next garbled circuit. This effectively emulates the execution of the same FMS MPC instance in the multi-round protocol, but without requiring any additional round. Now, the only thing left to address is how to communicate the correct labels.

Communicating the labels for each party's garbled circuit. The trick here is (again very similar to step 2) to tweak the circuit N, in that instead of outputting the two MPC_1 messages for the next level, N (with an additional random input r_i from each party i) now outputs labels corresponding to the messages.[8]

For security reasons, it is not possible to include the same randomness r_i in the input to each subsequent FMS instance. Thus, we use a carefully constructed tree-based PRF, following the GGM [GGM84] construction and pass along *not* the key of the PRF but a careful derivative that is sufficient for functionality and does not interfere with security.

Adaptivity in the choice of circuit. Our reusable two-round MPC protocol satisfies a strong adaptive security guarantee. In particular, the adversary may choose any circuit to compute after seeing the first round messages (and even after seeing the second round messages for other circuits computed on the same inputs). This stronger security is achieved based on the structure of our construction, since the first round messages of the FMS MPC used to compute the actual circuit are only revealed when the actual execution happens in the second round of the reusable protocol. In particular, we do not even have to rely on "adaptive" security of the underlying FMS protocol to achieve this property.[9]

3 Preliminaries

Let λ denote the security parameter. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for any polynomial $\mathsf{poly}(\cdot)$ there exists $\lambda_0 \in \mathbb{N}$ 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. We denote $[k]$ to be the set $\{1, \ldots, k\}$.

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

[8] Again, the actual protocol is slightly different, in that all labels are encrypted and sent along with the garbled circuits, and N outputs decryption keys corresponding to the correct labels.

[9] This is for reasons very similar to those in [DG17a].

Polynomial Time algorithm. We assume without loss of generality that the length of the random tape used by all cryptographic algorithms is λ.

3.1 Garbled Circuits

We recall the definition of a garbling scheme for circuits [Yao86] (see Applebaum et al. [AIK04], 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 $(\mathsf{Garble}, \mathsf{GEval})$. Garble is the circuit garbling procedure and GEval is the corresponding evaluation procedure. More formally:

- $(\tilde{\mathsf{C}}, \{\mathsf{lab}_{i,b}\}_{i\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{C}\right)$: Garble takes as input a security parameter 1^λ, a circuit C, and outputs a *garbled circuit* $\tilde{\mathsf{C}}$ along with labels $\mathsf{lab}_{i,b}$ where $i \in [n]$ (where n is the length of the input to C) and $b \in \{0,1\}$. Each label $\mathsf{lab}_{i,b}$ is assumed to be in $\{0,1\}^\lambda$.
- $y \leftarrow \mathsf{GEval}\left(\tilde{\mathsf{C}}, \{\mathsf{lab}_{i,x_i}\}_{i\in[n]}\right)$: Given a garbled circuit $\tilde{\mathsf{C}}$ and a sequence of input labels $\{\mathsf{lab}_{i,x_i}\}_{i\in[n]}$ (referred to as the garbled input), GEval outputs a string y.

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

$$\Pr\left[\mathsf{C}(x) = \mathsf{GEval}\left(\tilde{\mathsf{C}}, \{\mathsf{lab}_{i,x_i}\}_{i\in[n]}\right)\right] = 1$$

where $(\tilde{\mathsf{C}}, \{\mathsf{lab}_{i,b}\}_{i\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{C}\right)$.

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

$$\left(\tilde{\mathsf{C}}, \{\mathsf{lab}_{i,x_i}\}_{i\in[n]}\right) \overset{c}{\approx} \mathsf{Sim}\left(1^{|C|}, 1^{|x|}, \mathsf{C}(x)\right)$$

where $(\tilde{\mathsf{C}}, \{\mathsf{lab}_{i,b}\}_{i\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{C}\right)$ and $\overset{c}{\approx}$ denotes that the two distributions are computationally indistinguishable.

We use the notation of overline to denote a set of labels/keys for garbled circuits. For example, $\overline{\mathsf{lab}}$ refers to $\{\mathsf{lab}_{i,b}\}_{i\in[n],b\in\{0,1\}}$. In contrast, we reserve the notation of hat to denote a set where only one label/key is known per wire. For example, $\widehat{\mathsf{lab}}$ refers to $\{\mathsf{lab}_i\}_{i\in[n]}$.

3.2 Robust Private-Key Encryption

A robust private-key encryption scheme $(\mathsf{rob.enc}, \mathsf{rob.dec})$ satisfies the usual properties of correctness and semantic security, plus the following robustness property. For any PPT $\mathcal{A}$ and message m,

$$\Pr_{\substack{k\leftarrow\{0,1\}^\lambda,\\ c\leftarrow\mathsf{rob.enc}(k,m)}}[\mathsf{rob.dec}(k', c) \neq \perp : k' \leftarrow \mathcal{A}(c)] = \mathsf{negl}(\lambda).$$

A robust private-key encryption scheme can be constructed as follows. Let k be the key for a PRF $\mathsf{PRF}(k, \cdot) : \{0,1\}^\lambda \to \{0,1\}$. To encrypt a bit b, draw $r_i \leftarrow \{0,1\}^\lambda$ for each $i \in [2\lambda]$. Then draw $s \leftarrow \{0,1\}^\lambda$ and output $\{(r_i, \mathsf{PRF}(k, r_i))\}_{i\in[2\lambda]}, (s, \mathsf{PRF}(k, s) \oplus b)$ as the encryption of b. The decryption procedure on input ciphertext $\{(r_i, u_i)\}_{i\in[2\lambda]}, (s, v)$ and key k' will first check that $u_i = \mathsf{PRF}(k', r_i)$ for each (r_i, u_i). An adversary that can find a satisfying key k' will break security of the PRF, since with overwhelming probability, there will not exist such a k' relative to 2λ uniformly random pairs (r_i, u_i).

3.3 Two-Round MPC

Throughout this work, we will focus on two-round MPC protocols. We now define the syntax we follow for a two-round MPC protocol.

Definition 3.1 (Two-Round MPC Procotol)**.** An n-party two-round MPC protocol is described by a triplet of PPT algorithms $(\mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$ with the following syntax.

- $\mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, \mathsf{C}, i, x_i; r_i) =: (\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)})$: Takes as input 1^λ, a common random/reference string CRS, (the description of) a circuit C to be computed, identity of a party $i \in [n]$, input $x_i \in \{0,1\}^*$ and randomness $r_i \in \{0,1\}^\lambda$ (we drop mentioning the randomness explicitly when it is not needed). It outputs party i's first message $\mathsf{msg}_i^{(1)}$ and its private state $\mathsf{st}_i^{(1)}$.
- $\mathsf{MPC}_2(\mathsf{C}, \mathsf{st}_i^{(1)}, \{\mathsf{msg}_j^{(1)}\}_{j\in[n]}) \to (\mathsf{st}_i^{(2)}, \mathsf{msg}_i^{(2)})$: Takes as input (the description of) a circuit[10] C to be computed, the state[11] of a party $\mathsf{st}_i^{(1)}$, and the first round messages of all the parties $\{\mathsf{msg}_j^{(1)}\}_{j\in[n]}$. It outputs party i's second round message $\mathsf{msg}_i^{(2)}$ and its private state $\mathsf{st}_i^{(2)}$.
- $\mathsf{MPC}_3(\mathsf{st}_i^{(2)}, \{\mathsf{msg}_j^{(2)}\}_{j\in[n]}) =: y_i$: Takes as input the state of a party $\mathsf{st}_i^{(2)}$, and the second round messages of all the parties $\{\mathsf{msg}_j^{(2)}\}_{j\in[n]}$. It outputs the ith party's output y_i.

Each party runs the first algorithm MPC_1 to generate the first round message of the protocol, the second algorithm MPC_2 to generate the second round message of the protocol and finally, the third algorithm MPC_3 to compute the output. The messages are broadcasted after executing the first two algorithms, whereas the state is kept private.

Security. In this work we follow the standard real/ideal world paradigm for defining secure multi-party computation (MPC) as in [Gol04]. In order to distinguish it from the other enhanced notions, we will sometimes refer to this notion of two round MPC as the *vanilla* notion of two round MPC. We define this notion below for completeness. Parts of this section have been taken verbatim from [Gol04].

Consider n parties $P_1, \ldots, P_n$ with inputs $x_1, \ldots, x_n$ respectively that wish to interact in a protocol π to evaluate any circuit/functionality C on their joint inputs. The security of protocol π (with respect to a functionality C) is defined by comparing the real-world execution of the protocol with an ideal-world evaluation of f by a trusted party. Informally, it is required that for every adversary $\mathcal{A}$ that corrupts some subset of the parties $I \subset [n]$ and participates in the real execution of the protocol, there exist an adversary Sim, also referred to as a simulator, which can *achieve the same effect* in the ideal-world. Let's denote $\vec{x} = (x_1, \ldots, x_n)$. We now formally describe the security definition.

The real execution. In the real execution, the n-party protocol π for computing C is executed in the presence of an adversary $\mathcal{A}$. The adversary $\mathcal{A}$ takes as input the security parameter λ and an auxiliary input z. The honest parties follow the instructions of π. $\mathcal{A}$ sends all messages of the protocol on behalf of the corrupted parties (parties in I) following any arbitrary polynomial-time strategy.[12]

The interaction of $\mathcal{A}$ in the protocol π defines a random variable $\mathsf{REAL}_{\pi,\mathcal{A}}(\lambda, \vec{x}, z, I)$ whose value is determined by the coin tosses of the adversary and the honest parties. This random variable contains the output of the adversary (which may be an arbitrary function of its view) as well as the outputs of the honest parties.

[10] It might seem unnatural to include C in the input of MPC_2 when it was already used as an input for MPC_1. This is done to keep the notation consistent with a stronger notion of two-round MPC where C will be dropped from the input of MPC_1.

[11] Without loss of generality we may assume that the MPC_2 algorithm is deterministic given the state $\mathsf{st}_i^{(1)}$. Any randomness needed for the second round could be included in $\mathsf{st}_i^{(1)}$. Even in the reusable (defined later) case, it is possible to use a PRF computed on the input circuit to provide the needed randomness for the execution of MPC_2.

[12] A semi-honest adversary follows the protocol behaviour honestly.

The ideal execution. In the ideal execution, an ideal world adversary Sim interacts with a trusted party. The ideal execution proceeds as follows:

- **Send inputs to the trusted party:** Each honest party sends its input to the trusted party. Each corrupt party P_i, (controlled by Sim) may either send its input x_i or send some other input of the same length to the trusted party. Let x_i' denote the value sent by party P_i. Note that for a semi-honest adversary, $x_i' = x_i$ always.
- **Trusted party sends output to the adversary:** The trusted party computes $\mathsf{C}(x_1', \ldots, x_n') = (y_1, \ldots, y_n)$ and sends $\{y_i\}_{i \in I}$ to the adversary.
- **Adversary instructs trusted party to abort or continue:** This is formalized by having the adversary send either an abort or continue message to the trusted party. (A semi-honest adversary never aborts.) In the latter case, the trusted party sends to each honest party P_i its output value y_i. In the former case, the trusted party sends the special symbol $\perp$ to each honest party.
- **Outputs:** Sim outputs an arbitrary function of its view, and the honest parties output the values obtained from the trusted party.

The interaction of Sim with the trusted party defines a random variable $\mathsf{IDEAL}_{\mathsf{C},\mathsf{Sim}}(\lambda, \vec{x}, z, I)$. Having defined the real and the ideal worlds, we now proceed to define our notion of security.

Definition 3.2. *Let λ be the security parameter. Let C be an n-party functionality, and π be an n-party protocol for $n \in \mathsf{N}$. We say that π securely computes C in the presence of malicious (resp., semi-honest) adversaries if for every PPT real world adversary (resp., semi-honest adversary) $\mathcal{A}$, there exists a PPT ideal world adversary (resp., semi-honest adversary) Sim such that for any $\vec{x} = \{x_i\}_{i \in [n]} \in (\{0,1\}^*)^n$, $z \in \{0,1\}^*$, any $I \subset [n]$ and for any PPT distinugisher D, we have that*

$$|\Pr[D(\mathsf{REAL}_{\pi,\mathcal{A}}(\lambda, \vec{x}, z, I)) = 1] - \Pr[D(\mathsf{IDEAL}_{\mathsf{C},\mathsf{Sim}}(\lambda, \vec{x}, z, I)) = 1]|$$

is negligible in λ.

Remark 3.3 (Plain Model)**.** *We will also consider the setting of two-round MPC and its variants without the use of* CRS *(or, where* CRS *in the above algorithms is set to be $\perp$). We refer to such protocols as being in the* plain model*.*

3.3.1 First Message Succinct Two-Round MPC

We next define the notion of a *first message succinct* (FMS) two-round MPC protocol. This notion is a strengthening (in terms of efficiency) of the above described notion of (vanilla) two-round MPC. Informally, a two-round MPC protocol is first message succinct if the first round messages of all the parties can be computed without knowledge of the circuit being evaluated on the inputs. This allows parties to compute their first message independent of the circuit (in particular, independent also of its size) that will be computed in the second round.

Definition 3.4 (First Message Succinct Two-Round MPC)**.** *Let $\pi = (\mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$ be a two-round MPC protocol. Protocol π is said to be first message succinct if algorithm MPC_1 does not take as input the circuit C being computed. More specifically, it takes an input of the form $(1^\lambda, \mathsf{CRS}, i, x_i; r_i)$.*

Note that a first message succinct two-round MPC satisfies the same correctness and security properties as the (vanilla) two-round MPC.[13]

[13] In particular, for an FMS two-round MPC protocol, its first message is succinct but may not be reusable.

3.3.2 Reusable Two-Round MPC

We next define the notion of a *reusable* two-round MPC protocol, which can be seen as a strengthening of the security of a first message succinct two-round MPC protocol. Informally, reusability requires that the parties should be able to *reuse* the same first round message to securely evaluate an *unbounded* polynomially number of circuits $\mathsf{C}_1, \ldots, \mathsf{C}_\ell$, where ℓ is a polynomial (in λ) that is independent of any other parameter in the protocol. That is, for each circuit C_i, the parties can just run the second round of the protocol each time (using exactly the same first round messages) allowing the parties to evaluate the circuit on the *same inputs*. Note that each of these circuits can be of size an arbitrary polynomial in λ.

Very roughly, security requires that the transcript of all these executions along with the set of outputs should not reveal anything more than the inputs of the corrupted parties and the computed outputs.

We again formalize security (and correctness) via the real/ideal world paradigm. Consider n parties $P_1, \ldots, P_n$ with inputs $x_1, \ldots, x_n$ respectively. Also, consider an adversary $\mathcal{A}$ corrupting a set $I \subset [n]$ of parties.

The real execution. In the real execution, the n-party first message succinct two-round MPC protocol $\pi = (\mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$ is executed in the presence of an adversary $\mathcal{A}$. The adversary $\mathcal{A}$ takes as input the security parameter λ and an auxiliary input z. The execution proceeds in two phases:

- **Phase I:** All the honest parties $i \notin I$ execute the first round of the protocol by running the algorithm MPC_1 using their respective input x_i. They broadcast their first round message $\mathsf{msg}_i^{(1)}$ and preserve their secret state $\mathsf{st}_i^{(1)}$. Then the adversary $\mathcal{A}$ sends the first round messages on behalf of the corrupted parties following any arbitrary (polynomial-time computable) strategy (a semi-honest adversary follows the protocol behavior honestly and runs the algorithm $\mathsf{MPC}_1(\cdot)$).

- **Phase II (Reusable):** The adversary outputs a circuit C, which is provided to all parties.

  Next, each honest party computes the algorithm MPC_2 using this circuit C (and its secret state $\mathsf{st}_i^{(1)}$ generated as the output of MPC_1 in Phase I). Again, adversary $\mathcal{A}$ sends arbitrarily computed (in PPT) second round messages on behalf of the corrupt parties. The honest parties return the output of MPC_3 executed on their secret state and the received second round messages.

  The adversary $\mathcal{A}$ decides whether to continue the execution of a different computation. If yes, then the computation returns to the beginning of phase II. In the other case, phase II ends.

The interaction of $\mathcal{A}$ in the above protocol π defines a random variable $\mathsf{REAL}_{\pi,\mathcal{A}}(\lambda, \vec{x}, z, I)$ whose distribution is determined by the coin tosses of the adversary and the honest parties. This random variable contains the output of the adversary (which may be an arbitrary function of its view) as well as the output recovered by each honest party.

The ideal execution. In the ideal execution, an ideal world adversary Sim interacts with a trusted party. The ideal execution proceeds as follows:

1. **Send inputs to the trusted party:** Each honest party sends its input to the trusted party. Each corrupt party P_i, (controlled by Sim) may either send its input x_i or send some other input of the same length to the trusted party. Let x_i' denote the value sent by party P_i. Note that for a semi-honest adversary, $x_i' = x_i$ always.

2. **Adversary picks circuit:** Sim sends a circuit C to the ideal functionality which is also then forwarded to the honest parties.

3. **Trusted party sends output to the adversary:** The trusted party computes $\mathsf{C}(x_1', \ldots, x_n') = (y_1, \ldots, y_n)$ and sends $\{y_i\}_{i \in I}$ to the adversary.

4. **Adversary instructs trusted party to abort or continue:** This is formalized by having the adversary Sim send either a continue or abort message to the trusted party. (A semi-honest adversary never aborts.) In the latter case, the trusted party sends to each uncorrupted party P_i its output value y_i. In the former case, the trusted party sends the special symbol $\perp$ to each uncorrupted party.

5. **Reuse:** The adversary decides whether to continue the execution of a different computation. In the yes case, the ideal world returns to the start of Step 2.

6. **Outputs:** Sim outputs an arbitrary function of its view, and the honest parties output the values obtained from the trusted party.

Sim's interaction with the trusted party defines a random variable $\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, \vec{x}, z, I)$. Having defined the real and the ideal worlds, we now proceed to define our notion of security.

Definition 3.5. *Let λ be the security parameter. Let π be an n-party two-round protocol, for $n \in \mathsf{N}$. We say that π is a reusable two-round MPC protocol in the presence of malicious (resp., semi-honest) adversaries if for every PPT real world adversary (resp., semi-honest adversary) $\mathcal{A}$ there exists a PPT ideal world adversary (resp., semi-honest adversary) Sim such that for any $\vec{x} = \{x_i\}_{i\in[n]} \in (\{0,1\}^*)^n$, any $z \in \{0,1\}^*$, any $I \subset [n]$ and any PPT distinguisher $\mathcal{D}$, we have that*

$$|Pr[\mathcal{D}(\mathsf{REAL}_{\pi,\mathcal{A}}(\lambda, \vec{x}, z, I)) = 1] - Pr[\mathcal{D}(\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, \vec{x}, z, I)) = 1]|$$

is negligible in λ.

Remark 3.6 (Adaptivity of the circuits chosen.)**.** *Note that unlike the vanilla two-round MPC protocol (or even the first message succinct two-round MPC protocol), reusable two-round MPC allows the adversary to choose the circuits that the honest parties compute on* adaptively *based on its previous interactions with the honest parties.*

4 Step 1: Constructing Sharing-Compact HSS from HSS

In this section, we start by recalling the notion of homomorphic secret sharing (HSS) and defining our notion of *sharing-compact* HSS. We use the standard notion of HSS, which supports two parties and features additive reconstruction. In contrast, our notion of sharing compactness is for the multi-party case, but does not come with the typical bells and whistles of a standard HSS scheme — specifically, it features compactness *only* of the sharing algorithm and *without* additive reconstruction. For brevity, we refer to this notion of HSS as sharing-compact HSS (scHSS). In what follows, we give a construction of sharing-compact HSS and prove its security.

4.1 Homomorphic Secret Sharing

We start with a definition of HSS [BGI16]. The $\mathsf{HSS.Share}$ and $\mathsf{HSS.Eval}$ functionalities, along with the notion of semantic security, are standard, and taken from the paper [BGI16] introducing the primitive. The multi-evaluation correctness property we use is shown to be achievable in [BGI17]. Finally, we use the notion of error simulation from [BGI17], but we are slightly more explicit in the definition. In particular, we split $\mathsf{HSS.Share}$ into two parts $\mathsf{HSS.Share}^{(\mathsf{enc})}$ and $\mathsf{HSS.Share}^{(\mathsf{PRF})}$, and define an additional functionality $\mathsf{HSS.CheckEval}$, which in [BGI17] was essentially defined to be an auxiliary output of $\mathsf{HSS.Eval}$.

The following definition was shown in [BGI16, BGI17] to be achievable for program class $\mathcal{P}$ that contains circuits in NC^1, under the DDH assumption.

Definition 4.1 (Homomorphic Secret Sharing [BGI16, BGI17])**.** *A (2-party, $1-1/\mathsf{poly}$ correct, additive over $\mathbb{Z}_2$, multi-evaluation)* Homomorphic Secret Sharing *scheme for a class of programs $\mathcal{P}$ consists of algorithms (*$\mathsf{HSS.Share}, \mathsf{HSS.Eval}$*) with the following syntax:*

- $\mathsf{HSS.Share}(1^\lambda, x)$: *On input the security parameter* λ *and* $x \in \{0,1\}^n$*, the sharing algorithm outputs a pair of shares* $(\mathsf{sh}_0, \mathsf{sh}_1)$.
- $\mathsf{HSS.Eval}(\mathsf{id}, b, \mathsf{sh}, P, \delta)$: *On input identifier* id*, party index* $b \in \{0,1\}$*, share* sh*, program* $P \in \mathcal{P}$ *with* n *input bits and* m *output bits, and an error bound* $\delta > 0$*, the evaluation algorithm outputs* $y_b \in \{0,1\}^m$*, constituting party* b*'s part of the share of an output* $y \in \{0,1\}^m$.

The algorithm HSS.Share *is PPT, while* HSS.Eval *runs in time polynomial in its input and* $1/\delta$*. Now we continue with the correctness and security properties of* HSS.Share *and* HSS.Eval*. Semantic security is standard, and multi-evaluation correctness states that errors in evaluation are computationally indistinguishable from being independent events, over the randomness of* HSS.Share.

- ***Multi-evaluation correctness:*** *For every polynomial* m, s*, and nonuniform polynomial time distinguisher* $\mathcal{A}$*, there is a negligible function* ν *such that the following holds. For every positive integer* λ*, input* $x \in \{0,1\}^n$*, programs* $P_1, \ldots, P_{m(\lambda)} \in \mathcal{P}$ *of size* $s(\lambda)$ *with input length* n*, error bound* $\delta > 0$*, and distinct identifiers* $\mathsf{id}_1, \ldots, \mathsf{id}_{m(\lambda)} \in \{0,1\}^\lambda$*, there are error probabilities* $p_1, \ldots, p_{m(\lambda)} \leq \delta$ *such that the advantage of* $\mathcal{A}$ *in distinguishing between the outputs of the following two experiments is at most* $\nu(\lambda)$*:*
  - ***Experiment 1:*** *Output a bit sequence* $\tau_1, \ldots, \tau_{m(\lambda)}$ *where* $\Pr[\tau_i = 1] = p_i$ *and the* τ_i *are independent.*
  - ***Experiment 2:*** $(\mathsf{sh}_0, \mathsf{sh}_1) \leftarrow \mathsf{Share}(1^\lambda, x)$*. For* $i = 1, \ldots, m(\lambda)$*, and* $b \in \{0,1\}$*, let* $y_b^i \leftarrow \mathsf{Eval}(\mathsf{id}_i, b, \mathsf{sh}_b, P_i, \delta)$*, output 0 if* $y_0^i \oplus y_1^i = P_i(x)$ *and 1 otherwise.*
- ***Semantic security:*** *For any* $b \in \{0,1\}$*, pair of polynomial-length input sequences* $v_1, v_2, \ldots$ *and* $w_1, w_2, \ldots$ *such that* $|v_i| = |w_i|$*, and non-uniform polynomial time distinguisher* $\mathcal{A}$*, there is a negligible function* ν *such that for every positive integer* λ*,* $|\Pr[\mathcal{A}(V_\lambda^b) = 1] - \Pr[\mathcal{A}(W_\lambda^b) = 1]| \leq \nu(\lambda)$*, where* V_λ^b *(resp.* W_λ^b*) is obtained by letting* $(\mathsf{sh}_0, \mathsf{sh}_1) \leftarrow \mathsf{HSS.Share}(1^\lambda, v_\lambda)$ *(resp.* $(\mathsf{sh}_0, \mathsf{sh}_1) \leftarrow \mathsf{HSS.Share}(1^\lambda, w_\lambda)$*) and outputting* sh_b.

Now we discuss the error simulation. First, HSS.Share *may be split into two algorithms* $\mathsf{HSS.Share}^{(\mathsf{enc})}$ *and* $\mathsf{HSS.Share}^{(\mathsf{PRF})}$*, where* $\mathsf{HSS.Share}^{(\mathsf{enc})}(1^\lambda, x)$ *outputs* $(\widehat{\mathsf{sh}}_0, \widehat{\mathsf{sh}}_1)$ *and* $\mathsf{HSS.Share}^{(\mathsf{PRF})}(1^\lambda)$ *outputs* κ*, and* $\mathsf{sh}_0 \coloneqq (\widehat{\mathsf{sh}}_0, \kappa)$*,* $\mathsf{sh}_1 \coloneqq (\widehat{\mathsf{sh}}_1, \kappa)$*. Moreover, one can define the following algorithm* HSS.CheckEval.[14]

- $\mathsf{HSS.CheckEval}(\mathsf{id}, b, \mathsf{sh}, P, \delta)$: *On input identifier* id*, party index* $b \in \{0,1\}$*, share* sh*, program* $P \in \mathcal{P}$ *with* n *input bits, and an error bound* $\delta > 0$*, the check evaluation algorithm outputs a bit* $c \in \{0,1\}$.

This algorithms satisfies the following properties. In words, error simulation states that if HSS.CheckEval *outputs 1 on some party* b*'s share* sh_b *and program* P*, then party* b *is guaranteed that the* HSS *reconstruction will be correct. Correctness states that* HSS.CheckEval *will output 1 with high probability, and moreover that this probability depends only on the randomness of* $\mathsf{HSS.Share}^{(\mathsf{PRF})}$.

- ***Error simulation:*** *For any positive integer* λ*, input* $x \in \{0,1\}^n$*, bit* $b \in \{0,1\}$*, program* $P \in \mathcal{P}$ *with input length* n*, error bound* $\delta > 0$*, and identifier* $\mathsf{id} \in \{0,1\}^\lambda$*,*

$$\Pr\left[\begin{array}{c|l} \mathsf{HSS.CheckEval}(\mathsf{id}, b, \mathsf{sh}_b, P, \delta) = 1 & (\mathsf{sh}_0, \mathsf{sh}_1) \leftarrow \mathsf{HSS.Share}(1^\lambda, x) \\ \wedge\ y_0 \oplus y_1 \neq P(x) & y_0 \leftarrow \mathsf{HSS.Eval}(\mathsf{id}, 0, \mathsf{sh}_0, P, \delta) \\ & y_1 \leftarrow \mathsf{HSS.Eval}(\mathsf{id}, 1, \mathsf{sh}_1, P, \delta) \end{array}\right] = 0.$$

- ***Correctness of*** HSS.CheckEval***:*** *There is a negligible* ν *such that for any positive integer* λ*, input* $x \in \{0,1\}^n$*, bit* $b \in \{0,1\}$*, program* $P \in \mathcal{P}$ *with input length* n*, error bound* $\delta > 0$*, identifier* $\mathsf{id} \in \{0,1\}^\lambda$*, and pair of shares* $(\widehat{\mathsf{sh}}_0, \widehat{\mathsf{sh}}_1)$ *in the support of* $\mathsf{HSS.Share}^{(\mathsf{enc})}(1^\lambda, x)$*,*

$$\Pr_{\kappa \leftarrow \mathsf{HSS.Share}^{(\mathsf{PRF})}(1^\lambda)}[\mathsf{HSS.CheckEval}(\mathsf{id}, b, (\widehat{\mathsf{sh}}_b, \kappa), P, \delta) = 1] \geq 1 - \delta - \nu(\lambda).$$

[14] In [BGI17], this algorithm was defined as part of HSS.Eval itself and so was not given a separate name.

4.2 Sharing-Compact Homomorphic Secret Sharing

We continue with our definition of sharing-compact HSS, which differs from HSS in various ways:

- we support sharing among an arbitrary number of parties (in particular, more than 2);
- we have a simulation-based security definition;
- we support a notion of *robustness*;
- we have negligible correctness error;
- our reconstruction procedure is not necessarily additive;
- we require security for only *one* evaluation.

We do preserve the property that the sharing algorithm, and in particular, the size of the shares, is independent of the size of the program to be computed.

Definition 4.2 (Sharing-compact Homomorphic Secret Sharing (scHSS))**.** *A* scHSS *scheme for a class of programs $\mathcal{P}$ is a triple of PPT algorithms (*Share,Eval,Dec*) with the following syntax:*

$\mathsf{Share}(1^\lambda, n, x)$ *Takes as input a security parameter 1^λ, a number of parties n, and a secret $x \in \{0,1\}^*$, and outputs shares $(x_1, \ldots, x_n)$.*

$\mathsf{Eval}(j, P, x_j)$*: Takes as input a party index $j \in [n]$, a program P, and share x_j, and outputs a string $y_j \in \{0,1\}^*$.*

$\mathsf{Dec}(y_1, \ldots, y_n)$*: Takes as input all evaluated shares $(y_1, \ldots, y_n)$ and outputs $y \in \{0,1\}^*$.*

The algorithms satisfy the following properties.

- ***Correctness:*** *For any program $P \in \mathcal{P}$ and secret x,*

$$\Pr\left[\mathsf{Dec}(y_1, \ldots, y_n) = P(x) : \begin{array}{r}(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(1^\lambda, x) \\ \forall j, y_j \leftarrow \mathsf{Eval}(j, P, x_j)\end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

- ***Robustness:*** *For any non-empty set of honest parties $H \subseteq [n]$, program $P \in \mathcal{P}$, secret x, and PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{l}\mathsf{Dec}(y_1, \ldots, y_n) \in \{P(x), \perp\} \\ \quad : \\ \quad\end{array}\begin{array}{r}(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(1^\lambda, x) \\ \forall j \in H, y_j \leftarrow \mathsf{Eval}(j, P, x_j) \\ \{y_j\}_{j \in [n] \setminus H} \leftarrow \mathcal{A}(\{x_j\}_{j \in [n] \setminus H}, \{y_j\}_{j \in H})\end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

- ***Security:*** *There exists a PPT simulator $\mathcal{S}$ such that for any program $P \in \mathcal{P}$, any secret x, and any set of honest parties $H \subseteq [n]$ we have that:*

$$\left\{\{x_i\}_{i \in [n] \setminus H}, \{y_i\}_{i \in H} : \begin{array}{r}(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(1^\lambda, n, x), \\ \forall i \in H, y_i \leftarrow \mathsf{Eval}(i, P, x_i)\end{array}\right\} \stackrel{c}{\approx} \left\{\mathcal{S}(1^\lambda, P, n, H, P(x))\right\}.$$

4.3 Conforming Protocol

In our construction, we need a modification of the notion of conforming MPC protocol from [GS18]. Consider an MPC protocol Φ between parties $P_1, \ldots, P_n$. For each $i \in [n]$, we let $x_i \in \{0,1\}^m$ denote the input of party P_i. We consider any random coins used by a party to be part of its input (we can assume each party uses at most λ bits of randomness, and expands as necessary with a PRF). A conforming protocol Φ is defined by functions inpgen, gen, post, and computation steps or what we call *actions* $\phi_1, \cdots \phi_T$. The protocol Φ proceeds in three stages: the input sharing stage, the computation stage, and the output stage. For those familiar with the notion of conforming protocol from [GS18, GIS18], we outline the differences here.

- We split their function pre into $(\mathsf{inpgen}, \mathsf{gen})$, where inpgen is universal, in the sense that it only depends on the input length m (and in particular, not the function to be computed).
- We explicitly maintain a single public global state st that is updated one bit at a time. Each party's private state is maintained implicitly via their random coins s_i chosen during the input sharing phase.
- We require the transcript (which is fixed by the value of st at the end of the protocol) to be *publicly decodable*.

Next, we give our description of a conforming protocol.

- **Input sharing phase**: Each party i chooses random coins $s_i \leftarrow \{0,1\}^\lambda$, computes $(w_i, r_i) \coloneqq \mathsf{inpgen}(x_i, s_i)$ where $w_i = x_i \oplus r_i$, and broadcasts w_i. Looking ahead to the proof of Lemma 4.3, we will take s_i to be the seed of a $\mathsf{PRF}(s_i, \cdot) : \{0,1\}^* \to \{0,1\}$.
- **Computation phase**: Let T be a parameter that depends on the circuit C to be computed. Each party sets the global public state

$$\mathsf{st} \coloneqq (w_1 \| 0^{T/n} \| w_2 \| 0^{T/n} \| \cdots \| w_n \| 0^{T/n}),$$

and generates their secret state $v_i \coloneqq \mathsf{gen}(i, s_i)$.[15] Let ℓ be the length of st or v_i (st and v_i will be of the same length). We will also use the notation that for index $f \in [\ell]$, $v_{i,f} \coloneqq \mathsf{gen}_f(i, s_i)$.

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

$$\gamma_t = \mathsf{NAND}(\mathsf{st}_f \oplus v_{i,f}, \mathsf{st}_g \oplus v_{i,g}) \oplus v_{i,h}$$

and broadcasts γ_t to every other party.

3. Every party updates st_h to the bit value γ_t 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 $h \neq h'$ (this ensures that no state bit is ever overwritten).

- **Output phase**: Denote by $\Gamma = (\gamma_1, \ldots, \gamma_T)$ the transcipt of the protocol, and output $\mathsf{post}(\Gamma)$.

Lemma 4.3. *For any input length m, there exists a function inpgen such that any n party MPC protocol Π (where each party has an input of length at most m) can be written as a conforming protocol $\Phi = (\mathsf{inpgen}, \mathsf{gen}, \mathsf{post}, \{\phi_t\}_{t \in T})$ while inheriting the correctness and the security of the original protocol.*

The proof of this lemma is very similar to the proof provided in [GS18], but since we have tweaked the notion of a conforming protocol in multiple ways, we give the modified proof below.

Proof. First we define the universal function inpgen, which takes as input an $x \in \{0,1\}^m$ and random coins $s \in \{0,1\}^\lambda$. It interprets s as the seed of a PRF $\mathsf{PRF}(s, \cdot) : \{0,1\}^* \to \{0,1\}$. Then it computes the string $r \in \{0,1\}^m$, where $r_j \coloneqq \mathsf{PRF}(s, (j, \mathsf{inp}))$ (inp is a special symbol), and outputs $(w, r) \coloneqq (r \oplus x, r)$.

Now, let Π be any n party MPC protocol for computing a circuit C, where the functionality is defined so that everybody receives the same output. Without loss of generality we assume that in each round of Π, *one* party broadcasts *one* bit that is obtained by computing a circuit on its input and the messages it has received so far from other parties. Note that this restriction can be easily enforced by increasing the round complexity of the protocol to the communication complexity of the protocol. To achieve a publicly decodable transcript, simply require that the first party to receive the output broadcast it in the next round.

[15] Technically, gen should also take the parameters n, T as input, but we leave these implicit.

Let the round complexity (and also communication complexity) of Π be p. In every round $r \in [p]$ of Π, one of the parties computes a circuit and broadcasts the output bit. Let the circuit computed in round r be C_r. Without loss of generality we assume that (i) there exists q such that for each $r \in [p]$, we have that $q = |C_r|$, (ii) each C_r is composed of just NAND gates with fan-in two, and (iii) each party sends an equal number of bits in the execution of Π. All three of these conditions can be met by adding dummy gates and dummy rounds of interaction. We are now ready to describe our transformed conforming protocol Φ. The protocol Φ will have $T = pq$ rounds.

Recall that the global state st before the beginning of the computation phase will be set to

$$\mathsf{st} \coloneqq (w_1 \| 0^{T/n} \| w_2 \| 0^{T/n} \| \cdots \| w_n \| 0^{T/n}).$$

Let $\ell = nm + T$ be the length of st. We now describe the function $\mathsf{gen}(i, s_i)$, which for party i will generate an ℓ-bit string v_i used to mask the values of its intermediate computations. It will only have potentially non-zero values at locations $(i-1)\ell/n+1, \ldots, i\ell/n$. First, it sets $v_{i,(i-1)\ell/n+1}, \ldots, v_{i,(i-1)\ell/n+m}$ to the value r_i computed during inpgen. Next, it computes the string $u_i \in \{0,1\}^{T/n}$ as follows. For each $j \in [T/n]$ that is not a multiple of q, it sets $u_j \coloneqq \mathsf{PRF}(s_i, j)$. For each $j \in [T/n]$ that is a multiple of q, it sets $u_j \coloneqq 0$. Finally, it sets $v_{i,(i-1)\ell/n+m+1}, \ldots, v_{i,i\ell/n}$ to u_i.

We are now ready to describe the actions $\phi_1, \cdots \phi_T$. For each $r \in [p]$, round r in Π is expanded into q actions in Φ — namely, actions $\{\phi_t\}_t$ where $t \in \{(r-1)q+1, \cdots, rq\}$. Let P_i be the party that computes the circuit C_r and broadcasts the output bit in round r of Π. We now describe the ϕ_t for $t \in \{(r-1)q+1, \cdots, rq\}$. For each t, we set $\phi_t = (i, f, g, h)$ where f and g are the locations in st that the $(t-(r-1)q)$'th gate of C_r is computed on. Moreover, we set h to be the first location in st among the locations $(i-1)\ell/n + m + 1$ to $i\ell/n$ that has previously not been assigned to an action. (Note that this is pq/n locations which is exactly equal to the number of bits computed and broadcast by P_i.) Recall from before that on the execution of ϕ_t, party P_i sets $\gamma_t \coloneqq \mathsf{NAND}(\mathsf{st} \oplus v_{i,f}, \mathsf{st} \oplus v_{i,g}) \oplus v_{i,h}$ and broadcasts γ_t to all parties. Then, st_h is set to be the value of γ_t. Note that each value $\mathsf{st} \oplus v_{i,f}$ is either a bit of party i's input, a bit of i's intermediate computation, or a bit of the public transcript.

Now we need to argue that Φ preserves the correctness and security properties of Π. Observe that Φ is essentially the same as the protocol Π except that in Φ some additional bits are sent. Specifically, in addition to the messages that were sent in Π, parties computing Φ send a one-time pad of their inputs, plus $q-1$ additional bits per every bit sent in Π. Crucially, the messages sent in Π are sent *in the clear* in Φ, due to the fact that $v_{i,(i-1)\ell/n+m+j} = 0$ for j a multiple of q. Thus, no information is dropped in Φ, so correctness follows. Finally, note that each of the extra bits sent in Φ is masked by a bit generated with a secret PRF key. This ensures that, assuming PRF security, Φ achieves the same security properties as Π. □

4.4 Our Construction

We describe a sharing-compact HSS scheme for sharing an input $x \in \{0,1\}^m$ among n parties.

Ingredients: We use the following ingredients in our construction.

- An $n\lambda$-party conforming MPC protocol Φ (for computing an arbitrary functionality) with functions $\mathsf{inpgen}, \mathsf{gen}$, and post.
- A homomorphic secret sharing scheme ($\mathsf{HSS.Share}, \mathsf{HSS.Eval}$) supporting evaluations of circuits in NC^1. To ease notation in the description of our protocol, we will generally leave the party index, identifier, and error parameter δ implicit. The party index will be clear from context, the identifier can be the description of the function to be evaluated, and the error parameter will be fixed once and for all by the parties.
- A garbling scheme for circuits ($\mathsf{Garble}, \mathsf{GEval}$).
- A robust private-key encryption scheme ($\mathsf{rob.enc}, \mathsf{rob.dec}$) as defined in Section 3.2.

- A PRF that can be computed in NC^1.

Theorem 4.4. *Assuming a semi-honest MPC protocol (with any number of rounds) that can compute any polynomial-size functionality, a homomorphic secret sharing scheme supporting evaluations of circuits in NC^1, and a PRF that can be computed in NC^1, there exists a sharing-compact homomorphic secret sharing scheme supporting the evaluation of any polynomial-size circuit.*

Notation: As explained in Section 2, our construction at a high level follows the template of [GS18] (which we refer to as the GS protocol). In the evaluation step of our construction, each party generates a sequence of garbled circuits, one for each action step of the conforming protocol. For each of these action steps, the garbled circuit of one party speaks and the garbled circuits of the rest listen. We start by describing three circuits that aid this process: (i) circuit F (described in Figure 1), which includes the HSS evaluations enabling the speaking/listening mechanism, (ii) circuit P^* (described in Figure 2) garbled by the speaking party, and (iii) circuit P (described in Figure 3) garbled by the listening party.

(i) Circuit F. The speaking garbled circuit and the listening garbled circuit need shared secrets for communication. Using HSS, F provides an interface for setting up these shared secrets. More specifically, consider a speaking party j^* and a listening party $j \neq j^*$ during action t. In our construction, the parties j, j^* will be provided with HSS shares of their secrets $\{s_j, s_{j^*}\}, \{k_j, k_{j^*}\}$. Note that the order of s_j and s_{j^*} in $\{s_j, s_{j^*}\}$ and the order of k_j and k_{j^*} in $\{k_j, k_{j^*}\}$ is irrelevant. All of the secret information used by party j^* in computation of its conforming protocol messages is based on s_{j^*}. Also, during action t, party j's garbled circuit will need to output encrypted labels for its next garbled circuit. Secret k_j is used to generate any keys needed for encrypting garbled circuit labels. Concretely, in the circuit G (used inside F), observe that s_{j^*} is used to perform the computation of γ, and k_j is used to compute the "difference value", explained below.

$\mathsf{F}[t, \alpha, \beta, p](\mathsf{sh})$

Input: sh.
Hardcoded: The action number $t \in [T]$, bits α, β, and index $p \in [\lambda]$.

1. Let $\phi_t = (j^*, f, g, h)$.
2. Let $\mathsf{G}[t, \alpha, \beta, p, f, g, h]$ be the circuit that on input $\{s_j, s_{j^*}\}, \{k_j, k_{j^*}\}$:
   (a) Set $\gamma := \mathsf{NAND}\left(\alpha \oplus \mathsf{gen}_f(j^*, s_{j^*}), \beta \oplus \mathsf{gen}_g(j^*, s_{j^*})\right) \oplus \mathsf{gen}_h(j^*, s_{j^*})$.
   (b) If $\gamma = 0$, output 0^λ, else output $\mathsf{PRF}\left(k_j, (t, \alpha, \beta, p)\right)$.

Output: $\mathsf{HSS.Eval}(\mathsf{sh}, \mathsf{G}[t, \alpha, \beta, p, f, g, h])$.

Figure 1: The Circuit F.

Both party j and party j^* can compute F on their individual share of $\{s_j, s_{j^*}\}, \{k_j, k_{j^*}\}$. They either obtain the same output value (in the case that party j^*'s message bit for the t^{th} action is 0) or they obtain outputs that differ by a pseudorandom *difference* value known only to party j (in the case that party j^*'s message bit for the t^{th} action is 1). This difference value is equal to $\mathsf{PRF}(k_j, (t, \alpha, \beta, p))$, where t, α, β and p denote various parameters of the protocol.

Next, we'll see how the circuit F enables communication between garbled circuits. In our construction the speaking party will just output the evaluation of F on its share (for appropriate choices of t, α, β and p). On the other hand, party j will encrypt the zero-label for its next garbled circuit using the output of the

evaluation of F on its share (for appropriate choices of t, α, β and p) and will encrypt the one-label for its next garbled circuit using the exclusive or of this value and the difference value. Observe that the output of the speaking circuit will be exactly the key used to encrypt the label corresponding to the bit sent by j^* in the t^{th} action.

Finally, we need to ensure that each circuit G evaluated under the HSS can be computed in NC^1. Observe that G essentially only computes $\mathsf{gen}_f(j^*, s_{j^*})$ evaluations and $\mathsf{PRF}(k_j, \cdot)$ evaluations. The proof of Lemma 4.3 shows that $\mathsf{gen}_f(j^*, s_{j^*})$ may be computed with a single PRF evaluation using key s_{j^*}. Thus, if we take each s_j, k_j to be keys for a PRF computable in NC^1, it follows that G will be in NC^1.

(ii) The Speaking Circuit P^*. The construction of the speaking circuit is quite simple. The speaking circuit for the party j^* corresponding to action t computes the updated global state and the bit γ sent out in action t. However, it must somehow communicate γ to the garbled circuit of each $j \neq j^*$. This effect is achieved by having P^* return the output of F (on relevant inputs as explained above). However, technical requirements in the security proof preclude party j^* from hard-coding its HSS share sh into P^*, and having P^* compute on this share. Thus, we instead hard-code the outputs of F on all relevant inputs. More specifically, we hard-code $\left\{z_{j,p}^{(\alpha,\beta)}\right\}_{\substack{\alpha,\beta\in\{0,1\},\\ j\in[n']\setminus\{j^*\},\\ p\in[\lambda]}}$, where $z_{j,p}^{(\alpha,\beta)}$ is obtained as the output $\mathsf{F}[t,\alpha,\beta,p](\mathsf{sh})$.

$$\mathsf{P}^*\left[j^*, \left\{z_{j,p}^{(\alpha,\beta)}\right\}_{\substack{\alpha,\beta\in\{0,1\},\\ j\in[n']\setminus\{j^*\},\\ p\in[\lambda]}}, (v_f, v_g, v_h), \overline{\mathsf{lab}}\right](\mathsf{st})$$

Input: st.

Hardcoded: A (virtual) party index j^*, a set of strings $\left\{z_{j,p}^{(\alpha,\beta)}\right\}_{\substack{\alpha,\beta\in\{0,1\},\\ j\in[n']\setminus\{j^*\},\\ p\in[\lambda]}}$, three bits (v_f, v_g, v_h), and a set of labels $\overline{\mathsf{lab}} = \{\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}\}_{k\in[\ell]}$.

1. Compute $\gamma \coloneqq \mathsf{NAND}(\mathsf{st}_f \oplus v_f, \mathsf{st}_g \oplus v_g) \oplus v_h$.
2. Set $\mathsf{st}_h \coloneqq \gamma$.

Output: $\left(\gamma, \left\{z_{j,p}^{(\mathsf{st}_f,\mathsf{st}_g)}\right\}_{\substack{j\in[n']\setminus\{j^*\},\\ p\in[\lambda]}}, \{\mathsf{lab}_{k,\mathsf{st}_k}\}_{k\in[\ell]}\right)$.

Figure 2: The Speaking Circuit P^*.

(iii) The Listening Circuit P. The construction of the listening circuit mirrors that of the speaking circuit. The listening circuit outputs the labels for all wires except the h^{th} wire that it is listening on. For the h^{th} wire, the listening circuit outputs encryptions of the two labels under two distinct keys, where one of them will be output by the speaking circuit during this action. As in the case of speaking circuits, for technical reasons in the proof, we cannot have the listening circuit compute these value but must instead hard-code them. More specifically, we hard code $\left\{z_{p,0}^{(\alpha,\beta)}, z_{p,1}^{(\alpha,\beta)}\right\}_{\substack{\alpha,\beta\in\{0,1\},\\ p\in[\lambda]}}$ where $z_{p,0}^{(\alpha,\beta)}$ is obtained as $\mathsf{F}[t,\alpha,\beta,p](\mathsf{sh})$ and $z_{p,1}^{(\alpha,\beta)}$ is obtained as $z_{p,0}^{(\alpha,\beta)} \oplus \mathsf{PRF}\left(k_j, (t,\alpha,\beta,p)\right)$.

The Construction Itself: The foundation of a sharing-compact HSS for evaluating circuit C is a conforming protocol Φ (as described earlier in Section 4.3) computing the circuit C. Very roughly (and the details will become clear as we go along), in our construction, the Share algorithm will generate secret shares of the input x for the n parties. Additionally, the share algorithm generates the first round GS MPC messages on behalf of each party. The Eval algorithm will roughly correspond to the generation of the second

$$\mathsf{P}\left[j, \left\{z_{p,0}^{(\alpha,\beta)}, z_{p,1}^{(\alpha,\beta)}\right\}_{\substack{\alpha,\beta\in\{0,1\},\\ p\in[\lambda]}}, (f,g,h), \overline{\mathsf{lab}}\right](\mathsf{st})$$

Input: st.

Hardcoded. A (virtual) party index j, set of strings $\left\{z_{p,0}^{(\alpha,\beta)}, z_{p,1}^{(\alpha,\beta)}\right\}_{\substack{\alpha,\beta\in\{0,1\},\\ p\in[\lambda]}}$, three indices (f,g,h), and a set of labels $\overline{\mathsf{lab}} = \{\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}\}_{k\in[\ell]}$.

Output: $\left(\left\{\mathsf{rob.enc}\left(z_{p,b}^{(\mathsf{st}_f,\mathsf{st}_g)}, \mathsf{lab}_{h,b}\right), \right\}_{b\in\{0,1\},p\in[\lambda]}, \{\mathsf{lab}_{k,\mathsf{st}_k}\}_{k\in[\ell]\setminus\{h\}}\right)$.

Figure 3: The Listening Circuit P.

round messages of the GS MPC protocol. Finally, the Dec algorithm will perform the reconstruction, which corresponds to the output computation step in GS after all the second round messages have been sent out.

Our construction will differ from the GS protocol in the sense that the first round MPC messages generated by the sharing algorithm are independent of the circuit to be computed (and its size).

The Sharing Algorithm: Because of the inverse polynomial error probability in HSS (hinted at in Section 2 and explained in the proof), we need to use an $n' = n\lambda$ (virtual) party protocol rather than just an n party protocol. Each of the n parties actually messages for λ virtual parties. Barring this technicality and given our understanding of what needs to be shared to enable the communication between garbled circuits, the sharing is quite natural.

On input x, the share algorithm generates a secret sharing of x (along with the randomness needed for the execution of Φ) to obtain a share x_j for each virtual party $j \in [n']$. In addition, two PRF keys s_j, k_j for each virtual party $j \in [n']$ are sampled. Now, the heart of the sharing algorithm is the generation of HSS shares of $\{s_j, s_{j'}\}, \{k_j, k_{j'}\}$ for every pair of $j \neq j' \in [n']$, which are then provided to parties j and j'. Specifically, the algorithm computes shares $\mathsf{sh}_j^{\{j,j'\}}$ and $\mathsf{sh}_{j'}^{\{j,j'\}}$ as the output of $\mathsf{HSS.Share}\left(1^\lambda, (\{s_j, s_{j'}\}, \{k_j, k_{j'}\})\right)$. Note that we generate only one set of shares for each j, j' and the ordering of j and j' is irrelevant (we use the set notation to signify this).

$\mathsf{Share}(1^\lambda, n, x)$:

1. Let $n' = n\lambda$, $m' = m + \lambda$, and $x_1 \coloneqq (z_1 \| \rho_1) \in \{0,1\}^{m'}, \ldots, x_{n'} \coloneqq (z_{n'} \| \rho_{n'}) \in \{0,1\}^{m'}$, where $z_1, \ldots, z_{n'}$ is an additive secret sharing of x, and each $\rho_i \in \{0,1\}^\lambda$ is uniformly random. The ρ_i are the random coins used by each party in the MPC protocol Π underlying the conforming protocol Φ.
2. For each $j \in [n']$:
   (a) Draw PRF keys $s_j, k_j \leftarrow \{0,1\}^\lambda$, so that $\mathsf{PRF}(s_j, \cdot) : \{0,1\}^* \to \{0,1\}$ and $\mathsf{PRF}(k_j, \cdot) : \{0,1\}^* \to \{0,1\}^\lambda$, where both of these pseudorandom functions can be computed by NC^1 circuits.
   (b) Compute $(w_j, r_j) \coloneqq \mathsf{inpgen}(x_j, s_j)$.
3. For each $j \neq j' \in [n']$, compute $\left(\mathsf{sh}_j^{\{j,j'\}}, \mathsf{sh}_{j'}^{\{j,j'\}}\right) \leftarrow \mathsf{HSS.Share}\left(1^\lambda, (\{s_j, s_{j'}\}, \{k_j, k_{j'}\})\right)$.
4. Let $\overline{\mathsf{sh}}_j = \left(x_j, s_j, k_j, \left\{\mathsf{sh}_j^{\{j,j'\}}\right\}_{j'\in[n']\setminus\{j\}}\right)$.
5. For each $i \in [n]$, output party i's share $\mathsf{sh}_i \coloneqq \left(\{w_j\}_{j\in[n']}, \left\{\overline{\mathsf{sh}}_j\right\}_{j\in[(i-1)\lambda+1,\cdots,i\lambda]}\right)$.

The Evaluation Algorithm: Observe that the sharing algorithm is independent of the conforming protocol Φ (and the circuit C to be computed), thus achieving sharing compactness. This is due to the fact that the function inpgen is *universal* for conforming protocols Φ (as explained in Section 4.3).

In contrast, the evaluation algorithm will emulate the entire protocol Φ. First, it will set the error parameter δ for HSS, depending on the protocol Φ. Then, each virtual party j (where each party controls λ virtual parties) generates a garbled circuit for each action of the conforming protocol. For each action, the speaking party uses the speaking circuit P^* and the rest of the parties use the listening circuit P.

$\mathsf{Eval}(i, \mathsf{C}, \mathsf{sh}_i)$**:**

1. Parse sh_i as $\left(\{w_j\}_{j\in[n']}, \left\{\overline{\mathsf{sh}}_j\right\}_{j\in[(i-1)\lambda+1,\cdots,i\lambda]}\right)$, let T be a parameter[16] of the conforming protocol Φ computing C, and set the HSS error parameter $\delta = 1/8\lambda^2 T$.
2. Set $\mathsf{st} := (w_1 \| 0^{T/n'} \| w_2 \| 0^{T/n'} \| \cdots \| w_{n'} \| 0^{T/n'})$.
3. For each $j \in [(i-1)\lambda + 1, \cdots, i\lambda]$, run the following procedure.

   $\mathsf{VirtualEval}(j, \mathsf{C}, \overline{\mathsf{sh}}_j)$:

   (a) Parse $\overline{\mathsf{sh}}_j$ as $\left(x_j, s_j, k_j, \left\{\mathsf{sh}_j^{\{j,j'\}}\right\}_{j'\in[n']\setminus\{j\}}\right)$.

   (b) Compute $v_j := \mathsf{gen}(j, s_j)$.

   (c) Set $\overline{\mathsf{lab}}^{j,T+1} := \left\{\mathsf{lab}_{k,0}^{j,T+1}, \mathsf{lab}_{k,1}^{j,T+1}\right\}_{k\in[\ell]}$ where for each $k \in [\ell]$ and $b \in \{0,1\}$, $\mathsf{lab}_{k,b}^{j,T+1} := 0^\lambda$.

   (d) For each t from T down to 1:

      i. Parse ϕ_t as (j^*, f, g, h).

      ii. If $j = j^*$, compute (where P^* is described in Figure 2 and F is described in Figure 1)

$$\mathsf{arg}_1 := \left\{\mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j^*}^{\{j^*,j\}}\right)\right\}_{\substack{j\in[n']\setminus\{j^*\},\\ \alpha,\beta\in\{0,1\},\\ p\in[\lambda]}}$$
$$\mathsf{arg}_2 := (v_{j^*,f}, v_{j^*,g}, v_{j^*,h})$$
$$\left(\widetilde{\mathsf{P}}^{j^*,t}, \overline{\mathsf{lab}}^{j^*,t}\right) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{P}^*\left[j^*, \mathsf{arg}_1, \mathsf{arg}_2, \overline{\mathsf{lab}}^{j^*,t+1}\right]\right).$$

      iii. If $j \neq j^*$, compute (where P is described in Figure 3 and F is described in Figure 1)

$$\mathsf{arg}_1 := \left\{\begin{array}{c}\mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_j^{\{j^*,j\}}\right),\\ \mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_j^{\{j^*,j\}}\right) \oplus \mathsf{PRF}\left(k_j, (t,\alpha,\beta,p)\right)\end{array}\right\}_{\substack{\alpha,\beta\in\{0,1\},\\ p\in[\lambda]}}$$
$$\mathsf{arg}_2 := (f, g, h)$$
$$\left(\widetilde{\mathsf{P}}^{j,t}, \overline{\mathsf{lab}}^{j,t}\right) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{P}\left[j, \mathsf{arg}_1, \mathsf{arg}_2, \overline{\mathsf{lab}}^{j,t+1}\right]\right).$$

   (e) Set $\overline{y}_j := \left(\left\{\widetilde{\mathsf{P}}^{j,t}\right\}_{t\in[T]}, \left\{\mathsf{lab}_{k,\mathsf{st}_k}^{j,1}\right\}_{k\in[\ell]}\right)$. Recall st was defined in step 2.

4. Output $y_i := \left\{\overline{y}_j\right\}_{j\in[(i-1)\lambda+1,\cdots,i\lambda]}$.

[16] Recall that T is the number of actions to be taken.

The Decoding Algorithm: The decoding algorithm is quite natural given what we have seen so far. Garbled circuits from each virtual party are executed sequentially, communicating among themselves. This results in an evaluation of the conforming protocol Φ and the final output can be computed using the post algorithm.

$\mathsf{Dec}(y_1, \ldots, y_n)$**:**

1. For each $i \in [n]$, parse y_i as $\left\{ \left(\left\{ \widetilde{\mathsf{P}}^{j,t} \right\}_{t\in[T]}, \left\{ \mathsf{lab}_k^{j,1} \right\}_{k\in[\ell]} \right) \right\}_{j\in[(i-1)\lambda+1,\cdots,i\lambda]}$.
2. For each $j \in [n']$, let $\widetilde{\mathsf{lab}}^{j,1} := \left\{ \mathsf{lab}_k^{j,1} \right\}_{k\in[\ell]}$.
3. For each t from 1 to T,
   (a) Parse ϕ_t as (j^*, f, g, h).
   (b) Compute $\left(\gamma_t, \left\{ z^*_{j,p} \right\}_{\substack{j\in[n']\setminus\{j^*\},\\ p\in[\lambda]}}, \widetilde{\mathsf{lab}}^{j^*,t+1} \right) := \mathsf{GEval}\left(\widetilde{\mathsf{P}}^{j^*,t}, \widetilde{\mathsf{lab}}^{j^*,t} \right)$.
   (c) For each $j \neq j^*$:
      i. Compute $\left(\{\mathsf{elab}_{p,0}, \mathsf{elab}_{p,1}\}_{p\in[\lambda]}, \left\{ \mathsf{lab}_k^{j,t+1} \right\}_{k\in[\ell]\setminus\{h\}} \right) := \mathsf{GEval}\left(\widetilde{\mathsf{P}}^{j,t}, \widetilde{\mathsf{lab}}^{j,t} \right)$.
      ii. If there exists $p \in [\lambda]$, such that $\mathsf{rob.dec}\left(z^*_{j,p}, \mathsf{elab}_{p,\gamma_t} \right) \neq \perp$, then set the result to $\mathsf{lab}_h^{j,t+1}$. If all λ decryptions give $\perp$, then output $\perp$ and abort.
      iii. Set $\widetilde{\mathsf{lab}}^{j,t+1} := \left\{ \mathsf{lab}_k^{j,t+1} \right\}_{k\in[\ell]}$.
4. Set $\Gamma = (\gamma_1, \ldots, \gamma_T)$ and output $\mathsf{post}(\Gamma)$.

4.5 Correctness

We begin by introducing some notation. Recall that each share sh_i allows each party to construct the same shared initial global state $\mathsf{st} := (w_1 \| 0^{T/n'} \| w_2 \| 0^{T/n'} \| \cdots \| w_{n'} \| 0^{T/n'})$. Then, if the conforming protocol is followed correctly, each action ϕ_t should update st in one location, writing a single bit γ_t where there was previously a zero. We define $\mathsf{st}^{(1)} := \mathsf{st}$, and let $\mathsf{st}^{(t)}$ be the updated global state directly before action t. Thus $\mathsf{st}^{(T+1)}$ denotes the final global state. Note that in the correct execution of the conforming protocol, for each $\phi_t = (j^*, f, g, h)$, $\gamma_t = \mathsf{NAND}(\mathsf{st}^{(t)} \oplus v_{j^*,f}, \mathsf{st}^{(t)} \oplus v_{j^*,g}) \oplus v_{j^*,h}$.

Correctness will follow by showing that during Dec, each party's t'th garbled circuit is evaluated on $\mathsf{st}^{(t)}$. To see why this suffices, note that if this were the case, then for each $\phi_t = (j^*, f, g, h)$, party j^*'s t'th garbled circuit would output the correct $\gamma_t = \mathsf{NAND}(\mathsf{st}^{(t)} \oplus v_{j^*,f}, \mathsf{st}^{(t)} \oplus v_{j^*,g}) \oplus v_{j^*,h}$, by definition of the program P^*. Then the $\Gamma = (\gamma_1, \ldots, \gamma_T)$ computed during Dec would correspond to the correct transcript of the conforming protocol, and $\mathsf{post}(\Gamma)$ would give the correct output.

We can show the above claim by induction. The base case is clear since each party constructs the same $\mathsf{st} = \mathsf{st}^{(1)}$ during step 2 of Eval, and then releases the appropriate garbled labels for its first garbled circuit in step 3.(e) of Eval. Now assume the claim holds through action $t-1$. Let $\phi_t = (j^*, f, g, h)$, and consider one "listening" party $j \neq j^*$ (the following argument may be repeated for each $j \neq j^*$).

We know that the input to j^*'s t'th garbled circuit will be $\mathsf{st}^{(t)}$. By definition of P^*, the output of j^*'s garbled circuit includes the labels for j's $t+1$'st garbled circuit corresponding to input $\mathsf{st}^{(t+1)}$. Moreover, the output also includes

$$\left\{ z_{j,p}^{(\mathsf{st}_f^{(t)}, \mathsf{st}_g^{(t)})} \right\}_{p\in[\lambda]} := \left\{ z_p^* \right\}_{p\in[\lambda]},$$

where each string z_p^* is an HSS share of $\mathsf{G}[t, \mathsf{st}_f^{(t)}, \mathsf{st}_g^{(t)}, p, f, g, h]$ applied to the shared secret $\{s_j, s_{j^*}\}, \{k_j, k_{j^*}\}$.

We also know that the input to party j's t'th garbled circuit is $\mathsf{st}^{(t)}$. By definition of P, the output of j's garbled circuit includes all the labels for j's $t+1$'st garbled circuit corresponding to input $\mathsf{st}^{(t+1)}$, except the label $\mathsf{lab}_{h,\gamma_t}^{j,t+1}$. Moreover, the output also includes

$$\begin{aligned}
&\left\{\mathsf{rob.enc}\left(z_{p,0}^{(\mathsf{st}_f,\mathsf{st}_g)}, \mathsf{lab}_{h,0}^{j,t+1}\right), \mathsf{rob.enc}\left(z_{p,1}^{(\mathsf{st}_f,\mathsf{st}_g)}, \mathsf{lab}_{h,1}^{j,t+1}\right)\right\}_{p\in[\lambda]} \\
&= \left\{\mathsf{rob.enc}\left(z_{p,0}^{(\mathsf{st}_f,\mathsf{st}_g)}, \mathsf{lab}_{h,0}^{j,t+1}\right), \mathsf{rob.enc}\left(z_{p,0}^{(\mathsf{st}_f,\mathsf{st}_g)} \oplus \mathsf{PRF}\left(k_j, (t, \mathsf{st}_f^{(t)}, \mathsf{st}_g^{(t)}, p)\right), \mathsf{lab}_{h,1}^{j,t+1}\right)\right\}_{p\in[\lambda]} \\
&:= \left\{\mathsf{rob.enc}\left(z_p, \mathsf{lab}_{h,0}^{j,t+1}\right), \mathsf{rob.enc}\left(z_p \oplus \mathsf{PRF}\left(k_j, (t, \mathsf{st}_f^{(t)}, \mathsf{st}_g^{(t)}, p)\right), \mathsf{lab}_{h,1}^{j,t+1}\right)\right\}_{p\in[\lambda]} \\
&:= \left\{\mathsf{elab}_{p,0}, \mathsf{elab}_{p,1}\right\}_{p\in[\lambda]},
\end{aligned}$$

where each z_p is the other HSS share of $\mathsf{G}[t, \mathsf{st}_f^{(t)}, \mathsf{st}_g^{(t)}, p, f, g, h]$ applied to $\{s_j, s_{j^*}\}, \{k_j, k_{j^*}\}$.

Now it suffices to show that party j will recover $\mathsf{lab}_{h,\gamma_t}^{j,t+1}$ in step 3.(c).(ii) of Dec. First note that the γ_t computed within the program F is correct, since, for any $f \in [\ell]$, $v_{j^*,f} = \mathsf{gen}_f(j^*, s_{j^*})$. Now assume for the moment that HSS evaluation is perfectly correct. Then by definition of F, in the case that $\gamma_t = 0$, for any $p \in [\lambda]$ we have that $z_p^* \oplus z_p = 0^\lambda$, meaning that $z_p^* = z_p$. Otherwise, if $\gamma_t = 1$, we have that $z_p^* \oplus z_p = \mathsf{PRF}\left(k_j, (t, \mathsf{st}_f^{(t)}, \mathsf{st}_g^{(t)}, p)\right)$, meaning that $z_p^* = z_p \oplus \mathsf{PRF}\left(k_j, (t, \mathsf{st}_f^{(t)}, \mathsf{st}_g^{(t)}, p)\right)$. Thus, for either value of γ_t, and any $p \in [\lambda]$, $\mathsf{Dec}(z_p^*, \mathsf{elab}_{p,\gamma_t}) = \mathsf{lab}_{h,\gamma_t}^{j,t+1}$.

Finally, we must deal with the fact that HSS evaluation can fail with probability $1/\delta$. We appeal to the multi-evaluation correctness of HSS (see Definition 4.1), which implies that any event that occurs with negligible probability when assuming errors in multiple evaluations are truly independent, will also occur with negligible probability in the real execution of multiple evaluations. Thus we show that correctness holds with overwhelming probability, when assuming HSS evaluation errors occur independently and with probability at most $1/\delta = 1/8\lambda^2 T$. The probability that all of the λ decryptions in step 3.(c).(ii) fail is at most $(1/\delta)^\lambda = \mathsf{negl}(\lambda)$. Since this step is computed $Tn' = \mathsf{poly}(\lambda)$ times, the overall probability of failure is $\mathsf{negl}(\lambda)$.

4.6 Security

We begin by introducing some notation. Let j be one of the $n' = n\lambda$ virtual parties, and recall that

$$\left\{\left\{\mathsf{sh}_{j'}^{\{j',j\}}, \mathsf{sh}_j^{\{j',j\}}\right\}\right\}_{j'\in[n']\setminus\{j\}}$$

is the set of HSS shares involving j. Note that this set can be written (see Definition 4.1) as

$$\left\{\left\{\left(\widehat{\mathsf{sh}}_{j'}^{\{j',j\}}, \kappa^{\{j',j\}}\right), \left(\widehat{\mathsf{sh}}_j^{\{j',j\}}, \kappa^{\{j',j\}}\right)\right\}\right\}_{j'\in[n']\setminus\{j\}}.$$

Now, the conforming protocol Φ and circuit C determine the set of actions $\phi_t = (j^*, f, g, h)$, which determine the entire set of functions $\mathsf{G}[t, \alpha, \beta, p, f, g, h]$ to be computed on the HSS shares involving j. Call this set G_j, and note that it contains at most $8\lambda T$ functions. We call virtual party j *simulatable* if the following holds:

$$\forall\ j' \in [n'] \setminus \{j\}, \mathsf{G} \in \mathsf{G}_j, \mathsf{HSS.CheckEval}\left(\left(\widehat{\mathsf{sh}}_{j'}^{\{j',j\}}, \kappa^{\{j',j\}}\right), \mathsf{G}, \delta\right) = 1,$$

where we leave the inputs id, b implicit. In other words, j is simulatable if, from the viewpoint of all parties except j, it is guaranteed that all HSS evaluations with j will be correct. Note that (see Definition 4.1), the event that party j is simulatable depends *only* on the randomness of $\kappa^{\{j',j\}} \leftarrow \mathsf{HSS.Share}^{(\mathsf{PRF})}(1^\lambda)$, for $j' \neq j$.

4.6.1 The Simulator

Recall that we need to show that there exists a PPT simulator $\mathcal{S}$ such that for any n, x, C, and set of honest parties $H \subseteq [n]$,

$$\left\{ \{\mathsf{sh}_i\}_{i\in[n]\setminus H}, \{y_i\}_{i\in H} : \begin{array}{c} (\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, n, x), \\ \forall i \in H, y_i \leftarrow \mathsf{Eval}(i, \mathsf{C}, \mathsf{sh}_i) \end{array} \right\} \overset{c}{\approx} \left\{ \mathcal{S}(1^\lambda, \mathsf{C}, n, H, \mathsf{C}(x)) \right\}.$$

The construction of our simulator is based on the fact that at least one party is always honest, which means that at least λ virtual parties will be honest. We argue that at least one of these λ parties will be simulatable (or, error-free in terms of HSS evaluations from the perspective of all other parties). Our simulator proceeds by guessing the identity of a simulatable party. If it is the case that no HSS errors will occur on this party's shares, then the simulation continues. Otherwise, the entire set of $\kappa^{\{j',j\}} \leftarrow \mathsf{HSS.Share}^{(\mathsf{PRF})}$ for each pair of parties j, j' is re-sampled. More formally, the simulator $\mathcal{S}$ operates as follows, where Sim_Φ is the simulator for the underlying conforming protocol Φ.

$\mathcal{S}(1^\lambda, \mathsf{C}, n, H, \mathsf{C}(x))$:

1. Let $n' = n\lambda$ and let H' be the set of virtual honest parties. Pick a uniformly random j_{Sim} from H'.
2. Pick each of $\{x_j\}_{j\neq j_{\mathsf{Sim}}}$ uniformly at random.
3. Run $\left(\{s_j\}_{j\neq j_{\mathsf{Sim}}}, w_{j_{\mathsf{Sim}}}, \Gamma\right) \leftarrow \mathsf{Sim}_\Phi\left(1^\lambda, \mathsf{C}, n', \{x_j\}_{j\neq j_{\mathsf{Sim}}}, \{j_{\mathsf{Sim}}\}, \mathsf{C}(x)\right)$, where s_j are the random coins of party j, $w_{j_{\mathsf{Sim}}}$ is the simulated first round message of the "honest" party j_{Sim}, and $\Gamma = (\gamma_1, \ldots, \gamma_T)$ is the simulated transcript corresponding to the computation phase of the conforming protocol.
4. For $j \neq j_{\mathsf{Sim}}$, let $(w_j, r_j) := \mathsf{inpgen}(j, s_j)$.
5. Generate PRF keys $\{k_j\}_{j\neq j_{\mathsf{Sim}}}$, and set $s_{j_{\mathsf{Sim}}}, k_{j_{\mathsf{Sim}}} = 0^\lambda$.
6. For each $j \neq j' \in [n']$, compute $\left(\mathsf{sh}_j^{\{j,j'\}}, \mathsf{sh}_{j'}^{\{j,j'\}}\right) \leftarrow \mathsf{HSS.Share}\left(1^\lambda, (\{s_j, s_{j'}\}, \{k_j, k_{j'}\})\right)$.
7. Until j_{Sim} is simulatable, re-sample the entire set $\left\{\kappa^{\{j,j'\}}\right\}_{j,j'}$ for each $j \neq j'$, or abort after λ failed attempts.
8. For $j \neq j_{\mathsf{Sim}}$, let $\overline{\mathsf{sh}}_j = \left(x_j, s_j, k_j, \left\{\mathsf{sh}_j^{\{j,j'\}}\right\}_{j'\in[n']\setminus\{j\}}\right)$.
9. For $i \in [n] \setminus H$, set $\mathsf{sh}_i := \left(\{w_j\}_{j\in[n']}, \{\overline{\mathsf{sh}}_j\}_{j\in[(i-1)\lambda+1,\cdots,i\lambda]}\right)$.
10. For $j \in H' \setminus \{j_{\mathsf{Sim}}\}$, let $\overline{y}_j = \mathsf{VirtualEval}(j, \mathsf{C}, \overline{\mathsf{sh}}_j)$.
11. Set $\mathsf{st}^{(T+1)} := (w_1 \| 0^{T/n'} \| w_2 \| 0^{T/n'} \| \cdots \| w_{n'} \| 0^{T/n'}) \oplus u_\Gamma$, where u_Γ is defined to be zeros except that for all $t \in [T]$, $(u_\Gamma)_h = \gamma_h$, where h is such that $\phi_t = (j^*, f, g, h)$.
12. Generate $\overline{y}_{j_{\mathsf{Sim}}}$ via the following procedure:
    (a) Set $\widetilde{\mathsf{lab}}^{j_{\mathsf{Sim}},T+1} := \{0^\lambda\}_{k\in[\ell]}$.
    (b) For each t from T down to 1:
        i. Parse ϕ_t as (j^*, f, g, h).
        ii. If $j_{\mathsf{Sim}} = j^*$,

$$\left(\widetilde{\mathsf{P}}^{j_{\mathsf{Sim}},t}, \widetilde{\mathsf{lab}}^{j_{\mathsf{Sim}},t}\right) \leftarrow \mathsf{GSim}\left(1^{|\mathsf{P}_*|}, 1^\ell, \left(\gamma_t, \{z_{j,p}\}_{\substack{j\in[n']\setminus\{j_{\mathsf{Sim}}\},\\ p\in[\lambda]}}, \widetilde{\mathsf{lab}}^{j_{\mathsf{Sim}},t+1}\right)\right),$$

        where,

- If $\gamma_t = 0$,
$$z_{j,p} \coloneqq \mathsf{F}[t, \mathsf{st}_f^{(T+1)}, \mathsf{st}_g^{(T+1)}, p]\left(\mathsf{sh}_j^{\{j_{\mathsf{Sim}}, j\}}\right).$$
- If $\gamma_t = 1$,
$$z_{j,p} \coloneqq \mathsf{F}[t, \mathsf{st}_f^{(T+1)}, \mathsf{st}_g^{(T+1)}, p]\left(\mathsf{sh}_j^{\{j_{\mathsf{Sim}}, j\}}\right) \oplus \mathsf{PRF}\left(k_j, (t, \mathsf{st}_f^{(T+1)}, \mathsf{st}_g^{(T+1)}, p)\right).$$

iii. If $j_{\mathsf{Sim}} \neq j^*$,
$$\left(\widetilde{\mathsf{P}}^{j_{\mathsf{Sim}}, t}, \widetilde{\mathsf{lab}}^{j_{\mathsf{Sim}}, t}\right) \leftarrow \mathsf{GSim}\left(1^{|\mathsf{P}|}, 1^{\ell}, \left(\{\mathsf{elab}_{p,0}, \mathsf{elab}_{p,1}\}_{p\in[\lambda]}, \{\mathsf{lab}_k^{j_{\mathsf{Sim}}, t+1}\}_{k\neq h}\right)\right),$$
where for $u_p \leftarrow \{0,1\}^{\lambda}$,
$$\mathsf{elab}_{p,\gamma_t} \coloneqq \mathsf{rob.enc}\left(\mathsf{F}[t, \mathsf{st}_f^{(T+1)}, \mathsf{st}_g^{(T+1)}, p]\left(\mathsf{sh}_{j^*}^{\{j^*, j_{\mathsf{Sim}}\}}\right), \mathsf{lab}_h^{j_{\mathsf{Sim}}, t+1}\right),$$
$$\mathsf{elab}_{p,1-\gamma_t} \coloneqq \mathsf{rob.enc}\left(u_p, 0^{\lambda}\right).$$

(c) Set $\overline{y}_{j_{\mathsf{Sim}}} \coloneqq \left(\left\{\widetilde{\mathsf{P}}^{j_{\mathsf{Sim}}, t}\right\}_{t\in[T]}, \widetilde{\mathsf{lab}}^{j_{\mathsf{Sim}}, 1}\right)$.

13. For $i \in H$, set $y_i \coloneqq \{\overline{y}_j\}_{j\in[(i-1)\lambda+1, \cdots, i\lambda]}$.
14. Output $\left\{\{\mathsf{sh}_i\}_{i\in[n]\setminus H}, \{y_i\}_{i\in H}\right\}$.

4.6.2 Indistinguishability

Hybrid_0: The real distribution.

Hybrid_1: Ensure that at least one of the virtual honest parties is simulatable. Let $n' = n\lambda$ be the set of virtual parties in the protocol, and let H' be the set of virtual honest parties corresponding to the real honest parties. First, choose a party j_{Sim} uniformly from H'. Then, re-sample the entire set $\{\kappa^{\{j,j'\}} \leftarrow \mathsf{HSS.Share}^{(\mathsf{PRF})}(1^{\lambda})\}_{j,j'}$ for each $j \neq j'$, until j_{Sim} is simulatable, or abort after λ failed attempts.

Now we show that Hybrid_0 and Hybrid_1 are indistinguishable. First note that in Hybrid_0, each virtual honest party is simulatable with identical probability, and these outcomes are determined only by the $\mathsf{HSS.Share}^{(\mathsf{PRF})}$ computations. In Hybrid_1, we are using rejection sampling to generate the $\mathsf{HSS.Share}^{(\mathsf{PRF})}$ outcomes conditioned on the event that j_{Sim} is simulatable. Since j_{Sim} is chosen uniformly from all H', the only way that this is distinguishable from Hybrid_0 is if there are *no* simulatable virtual honest parties in Hybrid_0.

Whether or not each party is simulatable is not quite independent, so we'll have to be slightly careful about bounding the probability of zero simulatable parties in Hybrid_0. Towards this, we say that a set $\mathcal{J}$ of parties is *mutually simulatable* if $\forall\ j_1 \neq j_2 \in \mathcal{J}, \mathsf{G} \in \mathsf{G}_{j_1} \cap \mathsf{G}_{j_2}$,
$$\mathsf{HSS.CheckEval}\left(\left(\widehat{\mathsf{sh}}_{j_1}^{\{j_1,j_2\}}, \kappa^{\{j_1,j_2\}}\right), \mathsf{G}, \delta\right) = \mathsf{HSS.CheckEval}\left(\left(\widehat{\mathsf{sh}}_{j_2}^{\{j_1,j_2\}}, \kappa^{\{j_1,j_2\}}\right), \mathsf{G}, \delta\right) = 1.$$

Observe that conditioned on a set $\mathcal{J} \subseteq [n']$ being mutually simulatable, the probability that each individual $j \in \mathcal{J}$ is simulatable is completely independent.

Now, note that there are always at least λ virtual honest parties. Consider splitting the λ virtual honest parties into $\sqrt{\lambda}$ groups of size $\sqrt{\lambda}$. Since each individual party participates in at most $8\lambda T$ HSS computations throughout the protocol, whether a particular group is mutually simulatable depends on at most $8\lambda^{1.5}T/n'$ $\mathsf{HSS.CheckEval}$ computations. Since each $\mathsf{HSS.CheckEval}$ outputs 0 with

probability at most $\delta = 1/8\lambda^2 T$, a union bound shows that each group is mutually simulatable except with probability $(1/\sqrt{\lambda}n')$. Each group being mutually simulatable is an independent event, so the probability of zero mutually simulatable groups is at most $(1/\sqrt{\lambda}n')^{\sqrt{\lambda}} = \mathsf{negl}(\lambda)$. Now, an individual party is not simulatable with probability at most $\delta 8\lambda T = 1/\lambda$, and within a mutually simulatable group, these events are independent. Thus, the probability of zero mutually simulatable parties is at most $\mathsf{negl}(\lambda) + (1/\lambda)^{\sqrt{\lambda}} = \mathsf{negl}(\lambda)$.

Hybrid_2 : Simulate j_{Sim}'s HSS evaluations. We alter steps d.(ii) and d.(iii) of $\mathsf{VirtualEval}(j_{\mathsf{Sim}}, \mathsf{C}, \overline{\mathsf{sh}}_{j_{\mathsf{Sim}}})$. In particular, we compute $\mathsf{VirtualEval}(j_{\mathsf{Sim}}, \mathsf{C}, \overline{\mathsf{sh}}_{j_{\mathsf{Sim}}})$ without computing on any of j_{Sim}'s HSS shares. For each $t \in [T]$, and $\phi_t = (j^*, f, g, h)$:

- If $j_{\mathsf{Sim}} = j^*$, then for each $j \in [n'] \setminus \{j^*\}, \alpha, \beta \in \{0,1\}, p \in [\lambda]$, replace $\mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j_{\mathsf{Sim}}}^{\{j_{\mathsf{Sim}},j\}}\right)$ with

$$\mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j}^{\{j_{\mathsf{Sim}},j\}}\right) \text{ if } \gamma_t = 0$$
$$\mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j}^{\{j_{\mathsf{Sim}},j\}}\right) \oplus \mathsf{PRF}(k_j, (t,\alpha,\beta,p)) \text{ if } \gamma_t = 1.$$

- If $j_{\mathsf{Sim}} \neq j^*$, then for each $\alpha, \beta \in \{0,1\}, p \in [\lambda]$, replace

$$\left\{ \begin{array}{c} \mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j_{\mathsf{Sim}}}^{\{j_{\mathsf{Sim}},j^*\}}\right), \\ \mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j_{\mathsf{Sim}}}^{\{j_{\mathsf{Sim}},j^*\}}\right) \oplus \mathsf{PRF}\left(k_{j_{\mathsf{Sim}}}, (t,\alpha,\beta,p)\right) \end{array} \right\}$$

with

$$\left\{ \begin{array}{c} \mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j^*}^{\{j_{\mathsf{Sim}},j^*\}}\right), \\ \mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j^*}^{\{j_{\mathsf{Sim}},j^*\}}\right) \oplus \mathsf{PRF}\left(k_{j_{\mathsf{Sim}}}, (t,\alpha,\beta,p)\right) \end{array} \right\} \text{ if } \gamma_t = 0$$
$$\left\{ \begin{array}{c} \mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j^*}^{\{j_{\mathsf{Sim}},j^*\}}\right) \oplus \mathsf{PRF}\left(k_{j_{\mathsf{Sim}}}, (t,\alpha,\beta,p)\right) \\ \mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j^*}^{\{j_{\mathsf{Sim}},j^*\}}\right) \end{array} \right\} \text{ if } \gamma_t = 1.$$

Since all HSS evaluations that include j_{Sim} are perfectly correct, it follows from the definition of F that the above switches are perfectly indistinguishable.

Hybrid_3 : Replace $k_{j_{\mathsf{Sim}}}$ with 0 in all of the HSS secrets involving party j_{Sim}. This is indistinguishable from the previous hybrid by semantic security of HSS. For this step to work, we require that the distribution at this point can be generated without computing on any of j_{Sim}'s HSS shares. For some party j, the reduction can choose the values $(s_j, s_{j_{\mathsf{Sim}}}, k_j, k_{j_{\mathsf{Sim}}})$, and will receive from the challenger j's share (with j_{Sim}) of either $(s_j, s_{j_{\mathsf{Sim}}}, k_j, k_{j_{\mathsf{Sim}}})$ or $(s_j, s_{j_{\mathsf{Sim}}}, k_j, 0)$. It can use this information to generate the rest of distribution. Also note that the re-sampling procedure described in Hybrid_1 does not rely on j_{Sim}'s shares. Thus, the entire distribution can be generated without any of j_{Sim}'s HSS shares.

Hybrid_4 : Replace all values $\mathsf{PRF}\left(k_{j_{\mathsf{Sim}}}, (t,\alpha,\beta,p)\right)$ with a uniformly random string. This is indistiguishable from the previous hybrid by security of the PRF. Now observe that in step (d).(iii) of $\mathsf{VirtualEval}$, for each $t = \phi(j^*, f, g, h)$, whenever $j_{\mathsf{Sim}} \neq j^*$, we can write the second hard-coded argument to P as

$$\left\{ \begin{array}{c} \mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j^*}^{\{j_{\mathsf{Sim}},j^*\}}\right), \\ u_{t,\alpha,\beta,p} \end{array} \right\}_{\substack{\alpha,\beta\in\{0,1\},\\ p\in[\lambda]}} \text{ if } \gamma_t = 0$$
$$\left\{ \begin{array}{c} u_{t,\alpha,\beta,p}, \\ \mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j^*}^{\{j_{\mathsf{Sim}},j^*\}}\right) \end{array} \right\}_{\substack{\alpha,\beta\in\{0,1\},\\ p\in[\lambda]}} \text{ if } \gamma_t = 1,$$

where each $u_{t,\alpha,\beta,p} \leftarrow \{0,1\}^\lambda$.

Hybrid_5 : Simulate all of j_{Sim}'s garbled circuits. Indistinguishability follows from a sequence of subhybrids $\{\mathsf{Hybrid}_{4.t.0}, \mathsf{Hybrid}_{4.t.1}\}_{t\in[T]}$, two for each action. At each step $\phi_t = (j^*, f, g, h)$, we simulate j_{Sim}'s garbled circuit in $\mathsf{Hybrid}_{4.t.0}$. Then, if $j_{\mathsf{Sim}} \neq j^*$, in $\mathsf{Hybrid}_{4.t.1}$ we replace $\left\{\mathsf{rob.enc}\left(u_{t,\mathsf{st}_f,\mathsf{st}_g,p}, \mathsf{lab}_{h,1-\gamma_t}^{j_{\mathsf{Sim}},t+1}\right)\right\}_{p\in[\lambda]}$ with $\left\{\mathsf{rob.enc}\left(u_{t,\mathsf{st}_f,\mathsf{st}_g,p}, 0^\lambda\right)\right\}_{p\in[\lambda]}$ in the output of j_{Sim}'s t'th garbled circuit. This second step follows from semantic security of $(\mathsf{rob.enc}, \mathsf{rob.dec})$.

Hybrid_6 : Replace $s_{j_{\mathsf{Sim}}}$ with 0 in all of the HSS secrets involving party j_{Sim}. This is indistinguishable from the previous hybrid by semantic security of HSS. We again use the fact that the distribution at this point can be generated without any of j_{Sim}'s HSS shares.

Hybrid_7 : Simulate Φ. This is doable since j_{Sim}'s randomness $s_{j_{\mathsf{Sim}}}$ is not used to generate the rest of the distribution.

At this point, generating the distribution no longer requires x. Since the first part of each party's input is an additive share of x, and the second part is uniform, the inputs $\{x_j\}_{j\neq j_{\mathsf{Sim}}}$ are uniformly distributed. The resulting distribution is exactly the simulator described last section.

4.7 Robustness

To show that our construction achieves robustness, we actually show that robustness holds against the *simulated* distribution described above. By indistinguishability of the simulated distribution and the real distribution, this establishes the claim. More specifically, say that there exists a secret x, circuit C, set of honest parties $H \subseteq [n]$, and an adversary $\mathcal{A}$ that breaks the robustness experiment. Then, the following must also hold, where $\mathcal{S}$ is the simulator from last section.

$$\Pr\left[\mathsf{Dec}(y_1,\ldots,y_n) \notin \{\mathsf{C}(x), \perp\} : \begin{array}{c}(\{x_i\}_{i\in[n]\setminus H}, \{y_i\}_{i\in[H]}) \leftarrow \mathcal{S}(1^\lambda, |\mathsf{C}|, n, H, \mathsf{C}(x)) \\ \{y_i\}_{i\in[n]\setminus H} \leftarrow \mathcal{A}(\{x_i\}_{i\in[n]\setminus H}, \{y_i\}_{i\in[H]})\end{array}\right] = 1/\mathsf{poly}(\lambda).$$

Observe that the only way for $\mathcal{A}$ to force a bad outcome is if one of its garbled circuits manages to convince one of j_{Sim}'s garbled circuits of a "wrong" γ_t value. Specifically, it must be the case that for some $\phi_t = (j^*, f, g, h)$ with $j_{\mathsf{Sim}} \neq j^*$, during step 3.(c).(ii) of Dec there is some $z^*_{j_{\mathsf{Sim}},p}$ such that $\mathsf{Dec}(z^*_{j_{\mathsf{Sim}},p}, \mathsf{elab}_{p,1-\gamma_t}) \neq \perp$, where $\left(\{\mathsf{elab}_{p,0}, \mathsf{elab}_{p,1}\}_{p\in[\lambda]}, \left\{\mathsf{lab}_k^{j_{\mathsf{Sim}},t+1}\right\}_{k\in[\ell]\setminus\{h\}}\right) := \mathsf{GEval}\left(\widetilde{\mathsf{P}}^{j_{\mathsf{Sim}},t}, \widetilde{\mathsf{lab}}^{j_{\mathsf{Sim}},t}\right)$. But each $\mathsf{elab}_{p,1-\gamma_t} = \mathsf{enc}(u_p, 0^\lambda)$, where u_p is a uniformly random key drawn independent of $\mathcal{A}$'s view. Thus, the robustness of $(\mathsf{rob.enc}, \mathsf{rob.dec})$ directly implies that $\mathcal{A}$ cannot produce a key k' such that $\mathsf{Dec}(k', \mathsf{elab}_{p,1-\gamma_t}) \neq \perp$.

5 Step 2: FMS MPC from Sharing-Compact HSS

In this section, we use a sharing-compact HSS scheme to construct a *first message succinct* two-round MPC protocol that securely computes any polynomial-size circuit. We refer to Section 2.2 for a high-level overview of the construction. For modularity of presentation, we begin by defining a label encryption scheme.

Label Encryption. This is an encryption scheme designed specifically for encrypting a grid of $2 \times \ell$ garbled input labels corresponding to a garbled circuit with input length ℓ. The encryption algorithm takes as input a $2 \times \ell$ grid of strings (labels) along with a $2 \times \ell$ grid of keys. It encrypts each label using each corresponding key, making use of a *robust* private-key encryption scheme $(\mathsf{rob.enc}, \mathsf{rob.dec})$, as defined in Section 3.2. It then randomly permutes each pair (column) of ciphertexts, and outputs the resulting $2 \times \ell$ grid. On the other hand, decryption only takes as input a set of ℓ keys, that presumably correspond to exactly one ciphertext per column, or, exactly one input to the garbled circuit. The decryption algorithm

uses the keys to decrypt exactly one label per column, with the robustness of $(\mathsf{rob.enc}, \mathsf{rob.dec})$ ensuring that indeed only one ciphertext per column is able to be decrypted. The random permutations that occur during encryption ensure that a decryptor will recover a valid set of input labels *without knowing* which input they actually correspond to. This will be crucial in our construction.

$\mathsf{LabEnc}(\overline{K}, \overline{\mathsf{lab}})$: On input a key $\overline{K} = \{K_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$ and $\overline{\mathsf{lab}} = \{\mathsf{lab}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$ (where $K_{i,b}, \mathsf{lab}_{i,b} \in \{0,1\}^\lambda$), LabEnc draws n random bits $b'_i \leftarrow \{0,1\}$ and outputs $\overline{\mathsf{elab}} = \{\mathsf{elab}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$, where $\mathsf{elab}_{i,b} \coloneqq \mathsf{rob.enc}(K_{i,b\oplus b'_i}, \mathsf{lab}_{i,b\oplus b'_i})$.

$\mathsf{LabDec}(\widehat{K}, \overline{\mathsf{elab}})$**:** On input a key $\widehat{K} = \{K_i\}_{i\in[\ell]}$ and $\overline{\mathsf{elab}} = \{\mathsf{elab}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$, for each $i \in [\ell]$ output $\mathsf{rob.dec}(K_i, \mathsf{elab}_{i,0})$ if it is not $\perp$ and $\mathsf{rob.dec}(K_i, \mathsf{elab}_{i,1})$ otherwise.

5.1 Our Construction

The starting point of our construction is the observation that the only shortcoming of a scHSS scheme is that the Share algorithm must be executed by a trusted party/dealer. However, no trusted party is available in the setting of a first message succinct two-round MPC. A natural idea to address this problem is to have the parties use a vanilla two-round MPC to emulate the trusted execution of the sharing algorithm. Unfortunately, this results in parties receiving their shares at the end of the second round. Thus, the final protocol would require an extra round of communication, in which each party evaluates on their individual share and broadcasts the result. In order to avoid this extra round, in the second round we have the parties additionally send a garbled circuit that computes their third round message. To make this work, we actually have the vanilla two-round MPC output *keys* that allow recovery of garbled input labels corresponding to each party's scHSS share. This avoids the need for the third round altogether.

In more detail, each party i chooses a random string r_i in the first round. This string r_i will be used as the key to a PRF that generates a $2 \times \ell$ grid of LabEnc keys $\overline{K} \coloneqq \{\mathsf{PRF}(r_i, (t, b))\}_{t\in[\ell],b\in\{0,1\}}$. In round two, to jointly compute a circuit C, each party i generates a garbled circuit that takes as input a scHSS share csh_i and outputs the evaluated share $Y_i \coloneqq \mathsf{Eval}(i, \mathsf{C}, \mathsf{csh}_i)$. It then encrypts the $2 \times \ell$ grid of input labels using keys $\overline{K}$.

The functionality D that is computed by the vanilla MPC takes as input the party inputs $\{x_i\}_{i\in[n]}$, as well as the set of random strings $\{r_i\}_{i\in[n]}$. It first generates a sharing of the concatenated inputs $(\mathsf{csh}_1, \ldots, \mathsf{csh}_n) \leftarrow \mathsf{Share}(1^\lambda, n, (x_1 || \ldots || x_n))$. Then, instead of outputting the resulting shares in the clear, for each $i \in [n]$ it outputs keys that allow all parties to recover the input labels corresponding to input csh_i for party i's garbled circuit. In particular, it uses r_i to output the values $\{\mathsf{PRF}(r_i, (t, \mathsf{csh}_i[t]))\}_{i\in[\ell]}$. Now, after the second round of the vanilla MPC is broadcast, parties can recover each set of keys. Then, everybody can use the resulting keys to decrypt party i's garbled labels corresponding to input csh_i, and evaluate party i's garbled circuit to obtain Y_i. Crucially, due to the properties of LabEnc, this process leaks *nothing* about csh_i to each party $j \neq i$. Once all the Y_i are recovered, the reconstruction algorithm Dec of the scHSS scheme can be run to recover the output of C.

We present the formal construction in Figure 4. It is given for functionalities C where every party receives the same output, which is without loss of generality. Throughout, we will denote by ℓ the length of each party's scHSS share. Note that the circuit D used by the construction is defined immediately after in Figure 5. Finally, $p[t]$ denotes the t'th bit of a string $p \in \{0,1\}^*$.

Theorem 5.1. *Let $\mathcal{X} \in \{$semi-honest in the plain model, semi-honest in the common random/reference string model, malicious in the common random/reference string model$\}$. Assuming a (vanilla) $\mathcal{X}$ two-round MPC protocol and a scHSS scheme for polynomial-size circuits, there exists an $\mathcal{X}$ first message succinct two-round MPC protocol.*

5.2 Simulator

Remark 5.2. *Before proceeding to the proof of Theorem 5.1, we discuss the following point about simulation. Our simulator $\mathsf{FMS.Sim}$ will make use of the simulator Sim for the underlying vanilla MPC protocol. For*

A first message succinct MPC protocol $(\mathsf{FMS.MPC}_1, \mathsf{FMS.MPC}_2, \mathsf{FMS.MPC}_3)$

Main Ingredients:

- A (vanilla) two-round MPC protocol $(\mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$.
- A scHSS scheme $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$.
- A garbled circuit scheme $(\mathsf{Garble}, \mathsf{GEval})$.
- A label encryption scheme $(\mathsf{LabEnc}, \mathsf{LabDec})$.

$\mathsf{FMS.MPC}_1(1^\lambda, \mathsf{CRS}, i, x_i)$**:**

1. Draw $r_i \leftarrow \{0,1\}^\lambda$, and compute $(\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, \mathsf{D}, i, (x_i, r_i))$.
2. Output $\mathsf{FMS.st}_i^{(1)} := (\mathsf{st}_i^{(1)}, r_i)$ and $\mathsf{FMS.msg}_i^{(1)} := \mathsf{msg}_i^{(1)}$.

$\mathsf{FMS.MPC}_2(\mathsf{C}, \mathsf{FMS.st}_i^{(1)}, \{\mathsf{FMS.msg}_j^{(1)}\}_{j\in[n]})$**:**

1. Compute $(\mathsf{st}_i^{(2)}, \mathsf{msg}_i^{(2)}) \leftarrow \mathsf{MPC}_2(\mathsf{D}, \mathsf{st}_i^{(1)}, \{\mathsf{msg}_j^{(1)}\}_{j\in[n]})$.
2. Compute $(\tilde{\mathsf{C}}, \overline{\mathsf{lab}}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Eval}(i, \mathsf{C}, \cdot))$.
3. Compute $\overline{\mathsf{elab}} \leftarrow \mathsf{LabEnc}(\overline{K}, \overline{\mathsf{lab}})$ where $\overline{K} = \{\mathsf{PRF}(r_i, (t,b))\}_{t\in[\ell], b\in\{0,1\}}$.
4. Output $\mathsf{FMS.st}_i^{(2)} := \mathsf{st}_i^{(2)}$ and $\mathsf{FMS.msg}_i^{(2)} := (\mathsf{msg}_i^{(2)}, \tilde{\mathsf{C}}, \overline{\mathsf{elab}})$.

$\mathsf{FMS.MPC}_3(\mathsf{FMS.st}_i^{(2)}, \{\mathsf{FMS.msg}_j^{(2)}\}_{j\in[n]})$**:**

1. Compute $\{\widehat{K}_j\}_{j\in[n]} := \mathsf{MPC}_3(\mathsf{st}_i^{(2)}, \{\mathsf{msg}_j^{(2)}\}_{j\in[n]})$.
2. For each $j \in [n]$:
   (a) Compute $\widehat{\mathsf{lab}}_j := \mathsf{LabDec}(\widehat{K}_j, \overline{\mathsf{elab}}_j)$.
   (b) Compute $Y_j := \mathsf{GEval}(\tilde{\mathsf{C}}_j, \widehat{\mathsf{lab}}_j)$.
3. Output $y := \mathsf{Dec}(Y_1, \ldots, Y_n)$.

Figure 4: A first message succinct MPC protocol $(\mathsf{FMS.MPC}_1, \mathsf{FMS.MPC}_2, \mathsf{FMS.MPC}_3)$

Circuit D

Input: $(x_1, r_1), \ldots, (x_n, r_n)$

1. $(\mathsf{csh}_1, \ldots, \mathsf{csh}_n) \leftarrow \mathsf{Share}(1^\lambda, n, (x_1 || \ldots || x_n))$.
2. For each $i \in [n]$:
   (a) For each $t \in [\ell]$, set $K_{i,t} := \mathsf{PRF}(r_i, (t, \mathsf{csh}_i[t]))$.
   (b) Set $\widehat{K}_i := \{K_{i,t}\}_{t\in[\ell]}$.

Output: $(\widehat{K}_1 \ldots \widehat{K}_n)$

Figure 5: The (randomized) circuit D

convenience in the proof, we will actually assume that Sim *is black-box, and moreover, that it is* straight-line. *That is,* Sim *only requires oracle access to its adversary $\mathcal{A}$, and moreover, it interacts with $\mathcal{A}$ by sending*

a message $\left(\mathsf{CRS}, \{\mathsf{msg}_i^{(1)}\}_{i\in H}\right)$*, waiting for a response, sending a message* $\left\{\mathsf{msg}_i^{(2)}\right\}_{i\in H}$*, and waiting for a response (and it queries its ideal functionality after receiving its first response from* $\mathcal{A}$*, and before sending its second message to* $\mathcal{A}$*). We claim that this assumption is not necessary for the proof to go through, and expect that one could write a proof assuming a simulator that uses its adversary in an arbitrary manner. Even so, there are known constructions of two-round MPC from DDH [GS18, BL18] that anyway have simulators satisfying the above assumptions, in both the semi-honest setting and the malicious setting with CRS.*

Furthermore, we also claim that these assumptions are without loss of generality *in the setting of two-round MPC. We give a brief and informal argument for why this is the case. First note that for the semi-honest case, simulation is always straight-line. On the other hand, for the malicious case we are necessarily in the common reference string model. Now note that any two-round MPC protocol implies a two-round OT protocol. Next, we observe that any secure two-round OT protocol (regardless of its simulator) necessarily satisfies some basic game-based properties that give a (weak) notion of two-round OT. Thus, we can use the transformation of Döttling et al. [DGH+19] to construct a two-round OT with straight-line simulation. This two-round OT, when plugged into the compiler of Garg and Srinivasan [GS18] yields a two-round MPC protocol with a straight-line simulator.*

Now we prove Theorem 5.1. Let I denote the set of corrupt parties and $H := [n] \setminus I$ denote the set of honest parties. The description of the simulator FMS.Sim is given below. This simulator will have access to its own ideal functionality $\mathcal{F}_{\mathsf{C}}$. It takes as input the security parameter, auxiliary input z, set of corrupt parties I, and corrupt party inputs $\{x_i\}_{i\in I}$, and makes use of the simulator Sim for the vanilla two-round MPC, the simulator GSim for the garbling scheme, and the sharing-compact HSS simulator $\mathsf{scHSS.Sim}$.

Before we start describing the simulator and the hybrids, we provide descriptions of two circuits D' and D'' that are modifications of the circuit D. These circuits will be used in the simulator and the hybrids. The changes relative to D are written in red.

Circuit D'

Input: $(\mathsf{csh}_1, r_1), \ldots, (\mathsf{csh}_n, r_n)$

1. Skip this step.
2. For each $i \in [n]$:
   (a) For each $t \in [\ell]$, set $K_{i,t} := \mathsf{PRF}(r_i, (t, \mathsf{csh}_i[t]))$.
   (b) Set $\widehat{K}_i := \{K_{i,t}\}_{t\in[\ell]}$.

Output: $(\widehat{K}_1 \ldots \widehat{K}_n)$

Figure 6: The circuit D'

Circuit D''

Input: $(\mathsf{csh}_1, r_1), \ldots, (\mathsf{csh}_n, r_n)$

1. Skip this step.
2. For each $i \in [n]$:
   (a)
   - If $i \in I$, for each $t \in [\ell]$, set $K_{i,t} := \mathsf{PRF}(r_i, (t || \mathsf{csh}_i[t]))$.
   - If $i \in H$, for each $t \in [\ell]$, sample $K_{i,t} \leftarrow \mathcal{R}(r_i, (t, \mathsf{csh}_i[t]))$ where $\mathcal{R}(r_i, \cdot)$ is a random function with the same input and output lengths as PRF.

   (b) Set $\widehat{K}_i := \{K_{i,t}\}_{t \in [\ell]}$.

Output: $(\widehat{K}_1 \ldots \widehat{K}_n)$

Figure 7: The circuit D''

Now we present our simulator FMS.Sim.:

$\mathsf{FMS.Sim}^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i \in I})$**:**

1. For each $i \in [n]$, draw $r_i \leftarrow \{0,1\}^\lambda$. For $i \in H$ set $r'_i = r_i$.
2. Execute Sim with inputs $(1^\lambda, \bot, I, \{x_i, r_i\}_{i \in I})$, and receive $\left(\mathsf{CRS}, \{\mathsf{msg}_i^{(1)}\}_{i \in H}\right)$, where r_i for $i \in I$ are sampled uniformly.
3. Send $\left(\mathsf{CRS}, \{\mathsf{msg}_i^{(1)}\}_{i \in H}, \right)$ to $\mathcal{A}$, which returns $\left\{\mathsf{msg}_i^{(1)}\right\}_{i \in I}$.
4. Continue the execution of Sim to obtain values $\{x'_i, r'_i\}_{i \in I}$ submitted by Sim to its ideal functionality.
   (a) Send $\{x'_j\}_{j \in I}$ as the inputs of the corrupted parties to the ideal functionality $\mathcal{F}_{\mathsf{C}}$ and receive out' as the output.
   (b) Compute $\{\{\mathsf{csh}_i\}_{i \in I}, \{Y_i\}_{i \in H}\} \leftarrow \mathsf{scHSS.Sim}(1^\lambda, \mathsf{Eval}(\mathsf{C}, \cdot), n, H, \mathsf{out}')$.
   (c) Skip.
   (d) Compute $\{\widehat{K}_j\}_{j \in [n]} := \mathsf{D}''(\{\mathsf{csh}_j\}_{j \in [n]}, \{r'_j\}_{j \in [n]})$ where for each $j \in H$, csh_j is set to be the zero string. Provide $\{\widehat{K}_j\}_{j \in [n]}$ to Sim as the output of its query to the ideal functionality.
   (e) Receive $\{\mathsf{msg}_i^{(2)}\}_{i \in H}$ from Sim on behalf of the honest parties.
5. For each $i \in H$,
   (a) $(\tilde{\mathsf{C}}_i, \{\mathsf{lab}'_{i,t}\}_{t \in [\ell]}) \leftarrow \mathsf{GSim}(1^{|\mathsf{Eval}(\mathsf{C}, \cdot)|}, 1^{|\mathsf{csh}_i|}, Y_i)$. Let $\overline{\mathsf{lab}}_i = \{\mathsf{lab}_{i,t,0}, \mathsf{lab}_{i,t,1}\}_{t \in [\ell]}$, where each $\mathsf{lab}_{i,t,0} := \mathsf{lab}_{i,t,1} := \mathsf{lab}'_{i,t}$.
   (b) Compute $\overline{\mathsf{elab}}_i \leftarrow \mathsf{LabEnc}(\overline{K}_i, \overline{\mathsf{lab}}_i)$ where $\overline{K}_i = \{\mathcal{R}(r_i, (t, b))\}_{t \in [\ell], b \in \{0,1\}}$.
6. Send $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}}_i\right\}_{i \in H}$ to $\mathcal{A}$, which returns $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}}_i\right\}_{i \in I}$.
7. Compute out as follows:
   (a) Provide $\{\mathsf{msg}_j^{(2)}\}_{j \in I}$ to Sim. If Sim outputs continue then continue. Otherwise set $\mathsf{out} := \bot$ and skip to next step.
   (b) For each $j \in I$

i. Compute $\widehat{\mathsf{lab}}_j := \mathsf{LabDec}(\widehat{K}_j, \overline{\mathsf{elab}}_j)$.

ii. Evaluate $Y_j := \mathsf{GEval}(\tilde{\mathsf{C}}_j, \widehat{\mathsf{lab}}_j)$.

(c) If $\mathsf{out}' = \mathsf{Dec}(Y_1, \ldots, Y_n)$ then send continue to the ideal functionality else send $\perp$.

8. Output the output of $\mathcal{A}$.

5.3 Hybrids

Now we prove the distributions $\mathsf{REAL}_{\pi,\mathcal{A}}(\lambda, \vec{x}, z, I)$ and $\mathsf{IDEAL}_{\mathsf{Sim}^{\mathcal{A}}}(\lambda, \vec{x}, z, I)$ are computationally indistinguishable. We prove this claim by considering a sequence of hybrids. The distribution $\mathsf{REAL}_{\pi,\mathcal{A}}(\lambda, \vec{x}, z, I)$ is the same as the distribution output by $\mathsf{Hyb}_0^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]})$, and the distribution $\mathsf{IDEAL}_{\mathsf{Sim}^{\mathcal{A}}}(\lambda, \vec{x}, z, I)$ is the same as the distribution output by $\mathsf{Hyb}_7^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]})$. We will give the proof for the case where $\mathcal{A}$ is assumed to be malicious. The semi-honest case is identical except that one of the hybrids (specifically $\mathsf{Hyb}_6^{\mathcal{A}}$) can be skipped. We will make remarks when the proof for the semi-honest case will differ.

- $\underline{\mathsf{Hyb}_0^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]}) :}$ This hybrid corresponds to the real world execution of the FMS protocol in the presence of the adversary $\mathcal{A}$. In this hybrid, messages for the honest parties are generated according to the protocol specifications. Messages of the malicious parties are generated by $\mathcal{A}$. Formally, this hybrid proceeds as follows:

  1. For each $i \in [n]$, draw $r_i \leftarrow \{0,1\}^\lambda$. For $i \in H$ set $r'_i = r_i$.
  2. Sample CRS from the prescribed distribution and for $i \in H$, compute
  $$\left(\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)}\right) := \mathsf{MPC}_1\left(1^\lambda, \mathsf{CRS}, \mathsf{D}, i, (x_i, r_i)\right).$$
  3. Send $\left(\mathsf{CRS}, \{\mathsf{msg}_i^{(1)}\}_{i\in H}, \right)$ to $\mathcal{A}$, which returns $\left\{\mathsf{msg}_i^{(1)}\right\}_{i\in I}$.
  4. For $i \in H$, compute $\left(\mathsf{st}_i^{(2)}, \mathsf{msg}_i^{(2)}\right) \leftarrow \mathsf{MPC}_2\left(\mathsf{D}, \mathsf{st}_i^{(1)}, \{\mathsf{msg}_j^{(1)}\}_{j\in[n]}\right)$.
  5. For each $i \in H$,
     (a) Compute $(\tilde{\mathsf{C}}_i, \overline{\mathsf{lab}}_i) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Eval}(i, \mathsf{C}, \cdot))$.
     (b) Compute $\overline{\mathsf{elab}_i} \leftarrow \mathsf{LabEnc}(\overline{K}_i, \overline{\mathsf{lab}}_i)$ where $\overline{K}_i = \{\mathsf{PRF}(r_i, (t,b))\}_{t\in[\ell], b\in\{0,1\}}$.
  6. Send $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}}_i\right\}_{i\in H}$ to $\mathcal{A}$, which returns $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}}_i\right\}_{i\in I}$.
  7. Compute out as follows:
     (a) Compute $\{\widehat{K}_j\}_{j\in[n]} \coloneqq \mathsf{MPC}_3\left(\mathsf{st}_i^{(2)}, \{\mathsf{msg}_j^{(2)}\}_{j\in[n]}\right)$.
     (b) For each $j \in [n]$
        i. Compute $\widehat{\mathsf{lab}}_j := \mathsf{LabDec}(\widehat{K}_j, \overline{\mathsf{elab}}_j)$.
        ii. Evaluate $Y_j := \mathsf{GEval}(\tilde{\mathsf{C}}_j, \widehat{\mathsf{lab}}_j)$.
     (c) Set $\mathsf{out} := \mathsf{Dec}(Y_1, \ldots, Y_n)$.
  8. Output out and $\mathcal{A}$'s output.

- $\underline{\mathsf{Hyb}_1^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]}) :}$ In this hybrid, we change how the messages for the underlying vanilla two-round MPC are generated. More specifically, we use the simulator of the underlying vanilla two-round MPC to generate these messages, which affects Steps 2, 4 and 7. We give a formal description of this hybrid, and changes with respect to Hyb_0 are highlighted in red. We do this for every subsequent pair of hybrids. *Additionally, to avoid unnecessary repetition we only present a moving window snippet of the hybrids to which changes are being made.*

$$\vdots$$

2. Execute Sim with inputs $(1^\lambda, \perp, I, \{x_i, r_i\}_{i\in I})$, and receive $\left(\mathsf{CRS}, \{\mathsf{msg}_i^{(1)}\}_{i\in H}\right)$, where r_i for $i \in I$ are sampled uniformly.
3. Send $\left(\mathsf{CRS}, \{\mathsf{msg}_i^{(1)}\}_{i\in H}, \right)$ to $\mathcal{A}$, which returns $\left\{\mathsf{msg}_i^{(1)}\right\}_{i\in I}$.
4. Continue the execution of Sim to obtain values $\{x'_i, r'_i\}_{i\in I}$ queries by Sim to its ideal functionality.
   (a) For each $i \in H$ set $x'_i := x_i$ and $r'_i := r_i$. Set $\mathsf{out}' := \mathsf{C}(x'_1, \ldots x'_n)$.
   (b) Execute $(\mathsf{csh}_1, \ldots, \mathsf{csh}_n) \leftarrow \mathsf{Share}(1^\lambda, n, (x'_1 || \ldots || x'_n))$.
   (c) For each $j \in H$, set $Y_j := \mathsf{Eval}(i, \mathsf{C}, \mathsf{csh}_j)$.
   (d) Compute $\{\widehat{K}_j\}_{j\in[n]} := \mathsf{D}'(\{\mathsf{csh}_j\}_{j\in[n]}, \{r'_j\}_{j\in[n]})$ and provide it to Sim as the output of its query to the ideal functionality.
   (e) Next, Sim generates second round messages $\{\mathsf{msg}_i^{(2)}\}_{i\in H}$ on behalf of honest parties.

   Comment: Recall that D' is the circuit that does not compute Share but instead takes the output of Share as input.
5. For each $i \in H$,
   (a) Compute $(\tilde{\mathsf{C}}_i, \overline{\mathsf{lab}}_i) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Eval}(i, \mathsf{C}, \cdot))$.
   (b) Compute $\overline{\mathsf{elab}}_i \leftarrow \mathsf{LabEnc}(\overline{K}_i, \overline{\mathsf{lab}}_i)$ where $\overline{K}_i = \{\mathsf{PRF}(r_i, (t, b))\}_{t\in[\ell], b\in\{0,1\}}$.
6. Send $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}}_i\right\}_{i\in H}$ to $\mathcal{A}$, which returns $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}}_i\right\}_{i\in I}$.
7. Compute out as follows:
   (a) Provide $\{\mathsf{msg}_j^{(2)}\}_{j\in I}$ to Sim. If Sim outputs $\mathsf{continue}$ then continue. Otherwise set $\mathsf{out} := \perp$ and skip to next step.
   (b) For each $j \in I$
      i. Compute $\widehat{\mathsf{lab}}_j := \mathsf{LabDec}(\widehat{K}_j, \overline{\mathsf{elab}}_j)$.
      ii. Evaluate $Y_j := \mathsf{GEval}(\tilde{\mathsf{C}}_j, \widehat{\mathsf{lab}}_j)$.
   (c) Set $\mathsf{out} := \mathsf{Dec}(Y_1, \ldots, Y_n)$.

$\vdots$

$\mathsf{Hyb}_0^{\mathcal{A}} \approx_c \mathsf{Hyb}_1^{\mathcal{A}}$**.** The indistinguishability between the two hybrids follows directly based on the security of the underlying vanilla two-round MPC protocol.

- $\underline{\mathsf{Hyb}_2^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]})}$ **:** Observe that in hybrid $\mathsf{Hyb}_1^{\mathcal{A}}$, several calls to PRFs with keys $\{r_i\}_{i\in H}$ were made in the preparation of honest party messages. These calls are implicitly made in the execution of D' in Step 4d, and additionally in Step 5b. In this hybrid, instead of answering these queries as the output of pseudorandom functions, we answer using random functions $\mathcal{R}(r_i, \cdot)$. Note that we only replace the queries for $i \in H$ with random functions. Queries for $i \in I$ are still computed using a PRF.

$\vdots$

4. Continue the execution of Sim to obtain values $\{x'_i, r'_i\}_{i\in I}$ queries by Sim to its ideal functionality.
   (a) For each $i \in H$ set $x'_i := x_i$ and $r'_i := r_i$. Set $\mathsf{out}' := \mathsf{C}(x'_1, \ldots x'_n)$.
   (b) Execute $(\mathsf{csh}_1, \ldots, \mathsf{csh}_n) \leftarrow \mathsf{Share}(1^\lambda, n, (x'_1 || \ldots || x'_n))$.
   (c) For each $j \in H$, set $Y_j := \mathsf{Eval}(i, \mathsf{C}, \mathsf{csh}_j)$.
   (d) Compute $\{\widehat{K}_j\}_{j\in[n]} := \mathsf{D}''(\{\mathsf{csh}_j\}_{j\in[n]}, \{r'_j\}_{j\in[n]})$ and provide it to Sim as the output of its query to the ideal functionality.
   (e) Next, Sim generates second round messages $\{\mathsf{msg}_i^{(2)}\}_{i\in H}$ on behalf of honest parties.

Comment: Recall that D'' is same as circuit D' except that each call to $\mathsf{PRF}(r_i, \cdot)$ for $i \in H$ is answered as $\mathcal{R}(r_i, \cdot)$.

5. For each $i \in H$,
   (a) Compute $(\tilde{\mathsf{C}}_i, \overline{\mathsf{lab}}_i) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Eval}(i, \mathsf{C}, \cdot))$.
   (b) Compute $\overline{\mathsf{elab}}_i \leftarrow \mathsf{LabEnc}(\overline{K}_i, \overline{\mathsf{lab}}_i)$ where $\overline{K}_i = \{\mathcal{R}(r_i, (t, b))\}_{t \in [\ell], b \in \{0,1\}}$.

   Comment: Note that these $\overline{K}_i$ values include the $\widehat{K}_j$ values defined in Step 4d, since the same function $\mathcal{R}(r_j, \cdot)$ is used in both places.

$\vdots$

$\mathsf{Hyb}_1^{\mathcal{A}} \approx_c \mathsf{Hyb}_2^{\mathcal{A}}$. This indistinguishabilty follows directly from the security of PRF.

- $\underline{\mathsf{Hyb}_3^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i \in [n]})}$: In this hybrid, we remove information of one label per input wire, for each of the honest party garbled circuits. We do this by copying each label corresponding to the "correct" input. This change is done in Step 5 and will allow us to move to simulated garbled circuits in the next hybrid.

$\vdots$

4. Continue the execution of Sim to obtain values $\{x'_i, r'_i\}_{i \in I}$ queries by Sim to its ideal functionality.
   (a) For each $i \in H$ set $x'_i := x_i$ and $r'_i := r_i$. Set $\mathsf{out}' := \mathsf{C}(x'_1, \ldots x'_n)$.
   (b) Execute $(\mathsf{csh}_1, \ldots, \mathsf{csh}_n) \leftarrow \mathsf{Share}(1^\lambda, n, (x'_1 || \ldots || x'_n))$.
   (c) For each $j \in H$, set $Y_j := \mathsf{Eval}(i, \mathsf{C}, \mathsf{csh}_j)$.
   (d) Compute $\{\widehat{K}_j\}_{j \in [n]} := \mathsf{D}''(\{\mathsf{csh}_j\}_{j \in [n]}, \{r'_j\}_{j \in [n]})$ and provide it to Sim as the output of its query to the ideal functionality.
   (e) Next, Sim generates second round messages $\{\mathsf{msg}_i^{(2)}\}_{i \in H}$ on behalf of honest parties.
5. For each $i \in H$,
   (a) Compute $(\tilde{\mathsf{C}}_i, \overline{\mathsf{lab}}_i) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Eval}(i, \mathsf{C}, \cdot))$. For each $t \in [\ell]$, set $\mathsf{lab}_{i,t,1-\mathsf{csh}_i[t]} := \mathsf{lab}_{i,t,\mathsf{csh}_i[t]}$.
   (b) Compute $\overline{\mathsf{elab}}_i \leftarrow \mathsf{LabEnc}(\overline{K}_i, \overline{\mathsf{lab}}_i)$ where $\overline{K}_i = \{\mathcal{R}(r_i, (t, b))\}_{t \in [\ell], b \in \{0,1\}}$.

$\vdots$

$\mathsf{Hyb}_2^{\mathcal{A}} \approx_c \mathsf{Hyb}_3^{\mathcal{A}}$. This indistinguishabilty follows directly from the semantic security of LabEnc, which in turn follows from the semantic security of $\mathsf{rob.enc}$. Observe that each label $\mathsf{lab}_{i,t,1-\mathsf{csh}_i[t]}$ is encrypted under the key $\mathcal{R}(r_i, (t, 1 - \mathsf{csh}_i[t]))$, which is not used anywhere else. Furthermore, having the same labels for both input values (0 and 1) ensures that decrypting with any valid sequence of keys (in particular, one that is provided to Sim in Step 4d) would yield the correct sequence of labels $(\mathsf{lab}_{i,1,\mathsf{csh}_i[1]}, \ldots, \mathsf{lab}_{i,n,\mathsf{csh}_i[n]})$.

- $\underline{\mathsf{Hyb}_4^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i \in [n]})}$: In this hybrid, we change the share csh_j for each honest party j to be the zero string. This does not affect functionality because the same sequence of garbled labels will always be recovered no matter which key is used per input bit, since we are copying over the labels corresponding to the correct input.

$\vdots$

4. Continue the execution of Sim to obtain values $\{x'_i, r'_i\}_{i \in I}$ queries by Sim to its ideal functionality.
   (a) For each $i \in H$ set $x'_i := x_i$ and $r'_i := r_i$. Set $\mathsf{out}' := \mathsf{C}(x'_1, \ldots x'_n)$.

(b) Execute $(\mathsf{csh}_1, \ldots, \mathsf{csh}_n) \leftarrow \mathsf{Share}(1^\lambda, n, (x'_1 || \ldots || x'_n))$.

(c) For each $j \in H$, set $Y_j := \mathsf{Eval}(i, \mathsf{C}, \mathsf{csh}_j)$.

(d) Compute $\{\widehat{K}_j\}_{j\in[n]} := \mathsf{D}''(\{\mathsf{csh}_j\}_{j\in[n]}, \{r'_j\}_{j\in[n]})$ where for each $j \in H$ set csh_j is set to be the zero string and provide it to Sim as the output of its query to the ideal functionality.

(e) Next, Sim generates second round messages $\{\mathsf{msg}_i^{(2)}\}_{i\in H}$ on behalf of honest parties.

5. For each $i \in H$,

(a) Compute $(\tilde{\mathsf{C}}_i, \overline{\mathsf{lab}}_i) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Eval}(i, \mathsf{C}, \cdot))$. For each $t \in [\ell]$, set $\mathsf{lab}_{i,t,1-\mathsf{csh}_i[t]} := \mathsf{lab}_{i,t,\mathsf{csh}_i[t]}$.

(b) Compute $\overline{\mathsf{elab}}_i \leftarrow \mathsf{LabEnc}(\overline{K}_i, \overline{\mathsf{lab}}_i)$ where $\overline{K}_i = \{\mathcal{R}(r_i, (t, b))\}_{t\in[\ell], b\in\{0,1\}}$.

$\vdots$

$\mathsf{Hyb}_3^{\mathcal{A}} \equiv \mathsf{Hyb}_4^{\mathcal{A}}$. Recall that LabEnc randomly permutes each pair of labels corresponding to an input bit. Thus, which key is revealed for decryption does not affect the distribution, so this hybrid is perfectly indistinguishable from $\mathsf{Hyb}_3^{\mathcal{A}}$.

- $\underline{\mathsf{Hyb}_5^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]})}$: In this hybrid, we change how honest party garbled circuits are generated in Step 5. Specifically, we now generate the circuits using the simulator GSim.

$\vdots$

4. Continue the execution of Sim to obtain values $\{x'_i, r'_i\}_{i\in I}$ queries by Sim to its ideal functionality.

(a) For each $i \in H$ set $x'_i := x_i$ and $r'_i := r_i$. Set $\mathsf{out}' := \mathsf{C}(x'_1, \ldots x'_n)$.

(b) Execute $(\mathsf{csh}_1, \ldots, \mathsf{csh}_n) \leftarrow \mathsf{Share}(1^\lambda, n, (x'_1 || \ldots || x'_n))$.

(c) For each $j \in H$, set $Y_j := \mathsf{Eval}(i, \mathsf{C}, \mathsf{csh}_j)$.

(d) Compute $\{\widehat{K}_j\}_{j\in[n]} := \mathsf{D}''(\{\mathsf{csh}_j\}_{j\in[n]}, \{r'_j\}_{j\in[n]})$ and provide it to Sim as the output of its query to the ideal functionality.

(e) Next, Sim generates second round messages $\{\mathsf{msg}_i^{(2)}\}_{i\in H}$ on behalf of honest parties.

5. For each $i \in H$,

(a) Compute $(\tilde{\mathsf{C}}_i, \{\mathsf{lab}'_{i,t}\}_{t\in[\ell]}) \leftarrow \mathsf{GSim}(1^{|\mathsf{Eval}(\mathsf{C},\cdot)|}, 1^{|\mathsf{csh}_i|}, Y_i)$. Let $\overline{\mathsf{lab}}_i = \{\mathsf{lab}_{i,t,0}, \mathsf{lab}_{i,t,1}\}_{t\in[\ell]}$, where each $\mathsf{lab}_{i,t,0} := \mathsf{lab}_{i,t,1} := \mathsf{lab}'_{i,t}$.

(b) Compute $\overline{\mathsf{elab}}_i \leftarrow \mathsf{LabEnc}(\overline{K}_i, \overline{\mathsf{lab}}_i)$ where $\overline{K}_i = \{\mathcal{R}(r_i, (t, b))\}_{t\in[\ell], b\in\{0,1\}}$.

6. Send $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}}_i\right\}_{i\in H}$ to $\mathcal{A}$, which returns $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}}_i\right\}_{i\in I}$.

7. Compute out as follows:

(a) Compute $\{\widehat{K}_j\}_{j\in[n]} \coloneqq \mathsf{MPC}_3\left(\mathsf{st}_i^{(2)}, \{\mathsf{msg}_j^{(2)}\}_{j\in[n]}\right)$.

(b) For each $j \in [n]$

i. Compute $\widehat{\mathsf{lab}}_j := \mathsf{LabDec}(\widehat{K}_j, \overline{\mathsf{elab}}_j)$.

ii. Evaluate $Y_j := \mathsf{GEval}(\tilde{\mathsf{C}}_j, \widehat{\mathsf{lab}}_j)$.

(c) Set $\mathsf{out} := \mathsf{Dec}(Y_1, \ldots, Y_n)$.

$\vdots$

$\mathsf{Hyb}_4^{\mathcal{A}} \approx_c \mathsf{Hyb}_5^{\mathcal{A}}$. Indistinguishability follows directly from simulation security of the garbling scheme.

- $\underline{\mathsf{Hyb}_6^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]})}$: Skip this hybrid if in the semi-honest case. In this hybrid, we set of the honest party outputs to $\perp$ if the adversary can make them output the wrong output value.

$\vdots$

4. Continue the execution of Sim to obtain values $\{x'_i, r'_i\}_{i\in I}$ queries by Sim to its ideal functionality.
   (a) For each $i \in H$ set $x'_i := x_i$ and $r'_i := r_i$. Set $\mathsf{out}' := \mathsf{C}(x'_1, \ldots x'_n)$.
   (b) Execute $(\mathsf{csh}_1, \ldots, \mathsf{csh}_n) \leftarrow \mathsf{Share}(1^\lambda, n, (x'_1 || \ldots || x'_n))$.
   (c) For each $j \in H$, set $Y_j := \mathsf{Eval}(i, \mathsf{C}, \mathsf{csh}_j)$.
   (d) Compute $\{\widehat{K}_j\}_{j\in[n]} := \mathsf{D}''(\{\mathsf{csh}_j\}_{j\in[n]}, \{r'_j\}_{j\in[n]})$ and provide it to Sim as the output of its query to the ideal functionality.
   (e) Next, Sim generates second round messages $\{\mathsf{msg}_i^{(2)}\}_{i\in H}$ on behalf of honest parties.
5. For each $i \in H$,
   (a) Compute $(\tilde{\mathsf{C}}_i, \{\mathsf{lab}'_{i,t}\}_{t\in[\ell]}) \leftarrow \mathsf{GSim}(1^{|\mathsf{Eval}(\mathsf{C},\cdot)|}, 1^{|\mathsf{csh}_i|}, Y_i)$. Let $\overline{\mathsf{lab}_i} = \{\mathsf{lab}_{i,t,0}, \mathsf{lab}_{i,t,1}\}_{t\in[\ell]}$, where each $\mathsf{lab}_{i,t,0} := \mathsf{lab}_{i,t,1} := \mathsf{lab}'_{i,t}$.
   (b) Compute $\overline{\mathsf{elab}_i} \leftarrow \mathsf{LabEnc}(\overline{K}_i, \overline{\mathsf{lab}}_i)$ where $\overline{K}_i = \{\mathcal{R}(r_i, (t, b))\}_{t\in[\ell], b\in\{0,1\}}$.
6. Send $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}_i}\right\}_{i\in H}$ to $\mathcal{A}$, which returns $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}_i}\right\}_{i\in I}$.
7. Compute out as follows:
   (a) Provide $\{\mathsf{msg}_j^{(2)}\}_{j\in I}$ to Sim. If Sim outputs $\mathsf{continue}$ then continue. Otherwise set $\mathsf{out} := \perp$ and skip to next step.
   (b) For each $j \in I$
      i. Compute $\widehat{\mathsf{lab}}_j := \mathsf{LabDec}(\widehat{K}_j, \overline{\mathsf{elab}}_j)$.
      ii. Evaluate $Y_j := \mathsf{GEval}(\tilde{\mathsf{C}}_j, \widehat{\mathsf{lab}}_j)$.
   (c) If $\mathsf{out}' = \mathsf{Dec}(Y_1, \ldots, Y_n)$ then set $\mathsf{out} := \mathsf{C}(x'_1 \ldots x'_n)$ else set $\mathsf{out} := \perp$.

$\vdots$

$\mathsf{Hyb}_5^{\mathcal{A}} \approx_c \mathsf{Hyb}_6^{\mathcal{A}}$. This indinguishability follows from robustness of scHSS. To see why, note that the only possible way for the distributions output by $\mathsf{Hyb}_5^{\mathcal{A}}$ and $\mathsf{Hyb}_6^{\mathcal{A}}$ to differ is if $\mathsf{out} \neq \perp$ and $\mathsf{out} \neq \mathsf{out}'$ in $\mathsf{Hyb}_5^{\mathcal{A}}$. Observe that by definition, each $Y_j := \mathsf{GEval}(\tilde{\mathsf{C}}_j, \widehat{\mathsf{lab}}_j)$ for $j \in H$ computed in Step 7b of $\mathsf{Hyb}_5^{\mathcal{A}}$ is equal to the Y_j computed in step 4c by running the Share algorithm followed by Eval. Thus, producing $\{Y'_i\}_{i\in I}$ that combine with $\{Y_j\}_{j\in H}$ to cause Dec to output $y' \neq y$ implies an adversary that succeeds in the scHSS robustness experiment.

- $\underline{\mathsf{Hyb}_7^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]})}$: In this hybrid, we change how the sharing-compact HSS shares are generated. In particular, we use the $\mathsf{scHSS.Sim}$ to generate the $\{\mathsf{csh}\}_{i\in I}$ for the corrupted parties and $\{Y_i\}_{i\in H}$ for the corrupted parties.

$\vdots$

4. Continue the execution of Sim to obtain values $\{x'_i, r'_i\}_{i\in I}$ submitted by Sim to its ideal functionality.
   (a) For each $i \in H$ set $x'_i := x_i$ and $r'_i := r_i$. Set $\mathsf{out}' := \mathsf{C}(x'_1, \ldots x'_n)$.
   (b) Compute $\{\{\mathsf{csh}_i\}_{i\in I}, \{Y_i\}_{i\in H}\} \leftarrow \mathsf{scHSS.Sim}(1^\lambda, \mathsf{Eval}(\mathsf{C}, \cdot), n, H, \mathsf{out}')$.

(c) Skip.

(d) Compute $\{\widehat{K}_j\}_{j\in[n]} := \mathsf{D}''(\{\mathsf{csh}_j\}_{j\in[I]}, \{r_j\}_{j\in[n]})$ where for each $j \in H$, csh_j is set to be the zero string and provide it to Sim as the output of its query to the ideal functionality.

(e) Next, Sim generates second round messages $\{\mathsf{msg}_i^{(2)}\}_{i\in H}$ on behalf of honest parties.

5. For each $i \in H$,

(a) Compute $(\tilde{\mathsf{C}}_i, \{\mathsf{lab}'_{i,t}\}_{t\in[\ell]}) \leftarrow \mathsf{GSim}(1^{|\mathsf{Eval}(\mathsf{C},\cdot)|}, 1^{|\mathsf{csh}_i|}, Y_i)$. Let $\overline{\mathsf{lab}}_i = \{\mathsf{lab}_{i,t,0}, \mathsf{lab}_{i,t,1}\}_{t\in[\ell]}$, where each $\mathsf{lab}_{i,t,0} := \mathsf{lab}_{i,t,1} := \mathsf{lab}'_{i,t}$.

(b) Compute $\overline{\mathsf{elab}_i} \leftarrow \mathsf{LabEnc}(\overline{K}_i, \overline{\mathsf{lab}}_i)$ where $\overline{K}_i = \{\mathcal{R}(r_i, (t, b))\}_{t\in[\ell],b\in\{0,1\}}$.

7. Compute out as follows:

(a) Provide $\{\mathsf{msg}_j^{(2)}\}_{j\in I}$ to Sim. If Sim outputs $\mathsf{continue}$ then continue. Otherwise set $\mathsf{out} := \bot$ and skip to next step.

(b) For each $j \in I$

i. Compute $\widehat{\mathsf{lab}}_j := \mathsf{LabDec}(\widehat{K}_j, \overline{\mathsf{elab}}_j)$.

ii. Evaluate $Y_j := \mathsf{GEval}(\tilde{\mathsf{C}}_j, \widehat{\mathsf{lab}}_j)$.

(c) If $\mathsf{out}' = \mathsf{Dec}(Y_1, \ldots, Y_n)$ then set $\mathsf{out} := \mathsf{C}(x'_1 \ldots x'_n)$ else set $\mathsf{out} := \bot$.

8. Output out and $\mathcal{A}$'s output.

$\mathsf{Hyb}_6^{\mathcal{A}} \approx_c \mathsf{Hyb}_7^{\mathcal{A}}$. This indistinguishability follows directly from simulation security of the scHSS scheme.

- $\underline{\mathsf{Hyb}_8^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]})}$: In this hybrid, instead of computing the out' itself, we obtain it by querying the ideal functionality $\mathcal{F}_{\mathsf{C}}$.

$\vdots$

4. Continue the execution of Sim to obtain values $\{x'_i, r'_i\}_{i\in I}$ submitted by Sim to its ideal functionality.

(a) Send $\{x'_j\}_{j\in I}$ as the inputs of the corrupted parties to the ideal functionality $\mathcal{F}_{\mathsf{C}}$ and receive out' as the output.

(b) Compute $\{\{\mathsf{csh}_i\}_{i\in I}, \{Y_i\}_{i\in H}\} \leftarrow \mathsf{scHSS.Sim}(1^\lambda, \mathsf{Eval}(\mathsf{C},\cdot), n, H, \mathsf{out}')$.

(c) Skip.

(d) Compute $\{\widehat{K}_j\}_{j\in[n]} := \mathsf{D}''(\{\mathsf{csh}_j\}_{j\in[n]}, \{r'_j\}_{j\in[n]})$ where for each $j \in H$, csh_j is set to be the zero string. Provide $\{\widehat{K}_j\}_{j\in[n]}$ to Sim as the output of its query to the ideal functionality.

(e) Receive $\{\mathsf{msg}_i^{(2)}\}_{i\in H}$ from Sim on behalf of the honest parties.

5. For each $i \in H$,

(a) $(\tilde{\mathsf{C}}_i, \{\mathsf{lab}'_{i,t}\}_{t\in[\ell]}) \leftarrow \mathsf{GSim}(1^{|\mathsf{Eval}(\mathsf{C},\cdot)|}, 1^{|\mathsf{csh}_i|}, Y_i)$. Let $\overline{\mathsf{lab}}_i = \{\mathsf{lab}_{i,t,0}, \mathsf{lab}_{i,t,1}\}_{t\in[\ell]}$, where each $\mathsf{lab}_{i,t,0} := \mathsf{lab}_{i,t,1} := \mathsf{lab}'_{i,t}$.

(b) Compute $\overline{\mathsf{elab}_i} \leftarrow \mathsf{LabEnc}(\overline{K}_i, \overline{\mathsf{lab}}_i)$ where $\overline{K}_i = \{\mathcal{R}(r_i, (t, b))\}_{t\in[\ell],b\in\{0,1\}}$.

6. Send $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}_i}\right\}_{i\in H}$ to $\mathcal{A}$, which returns $\left\{\mathsf{msg}_i^{(2)}, \tilde{C}_i, \overline{\mathsf{elab}_i}\right\}_{i\in I}$.

7. Compute out as follows:

(a) Provide $\{\mathsf{msg}_j^{(2)}\}_{j\in I}$ to Sim. If Sim outputs $\mathsf{continue}$ then continue. Otherwise set $\mathsf{out} := \bot$ and skip to next step.

(b) For each $j \in I$

i. Compute $\widehat{\mathsf{lab}}_j := \mathsf{LabDec}(\widehat{K}_j, \overline{\mathsf{elab}}_j)$.

ii. Evaluate $Y_j := \mathsf{GEval}(\tilde{\mathsf{C}}_j, \widehat{\mathsf{lab}}_j)$.

(c) If $\mathsf{out}' = \mathsf{Dec}(Y_1, \ldots, Y_n)$ then send continue to the ideal functionality else send $\bot$.

8. Output the output of $\mathcal{A}$.

$\mathsf{Hyb}_7^{\mathcal{A}} \equiv \mathsf{Hyb}_8^{\mathcal{A}}$. By definition of the ideal functionality $\mathcal{F}_{\mathsf{C}}$, this is merely a syntactic change, and the two hybrids are equivalent.

6 Step 3: Two-Round Reusable MPC from FMS MPC

In this section, we present our compiler that takes any two-round FMS MPC and produces a two-round reusable MPC. We begin with some notation.

Notation. For a circuit C, let $\langle \mathsf{C} \rangle$ be a string obtained by concatenating the length of C (expanded to λ bits by padding) with C, i.e. $\langle \mathsf{C} \rangle := |\mathsf{C}| \,\|\, \mathsf{C} \in \{0,1\}^*$. Throughout, we will denote by ℓ the length of the string $(\mathsf{st}_j^{(1)}, \{\mathsf{msg}_j^{(1)}\}_{j\in[n]})$.

PRG. We will use a length-quadrupling pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{4\lambda}$. We write $\mathsf{G}_0, \mathsf{G}_1, \mathsf{H}_0, \mathsf{H}_1$ to denote the first, second, third and fourth part of the output. For example, for seed $r \in \{0,1\}^\lambda$, let $\mathsf{PRG}(r) = y_0 \| y_1 \| y_2 \| y_3 = \mathsf{G}_0(r) \| \mathsf{G}_1(r) \| \mathsf{H}_0(r) \| \mathsf{H}_1(r)$.

$\mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)$ Let $(\mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$ be a two-round MPC protocol. We will be garbling the circuit $\mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)$, which is the circuit computing the MPC_2 algorithm with the first input fixed to be the description of a circuit C and the other two inputs undefined (to be decided later).

Wrap. Consider a standard private-key encryption scheme $(\mathsf{enc}, \mathsf{dec})$. Suppose F is a function that takes an input x and returns $(y_1, y_2) := F(x)$ where y_1, y_2 are of specific lengths. Then we define a function $\mathsf{Wrap}(\mathsf{sk}, F)$ that on input $x \in \{0,1\}^*$, computes $(y_1, y_2) := F(x)$, and returns (z, y_2), where $z = \mathsf{enc}(\mathsf{sk}, y_1)$. Looking ahead, we will be using a circuit that implements $\mathsf{Wrap}(\mathsf{sk}, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot))$. In particular, the circuit $\mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)$ returns a state and message pair $(\mathsf{st}, \mathsf{msg})$ and then Wrap encrypts the state st but outputs msg in the clear.

6.1 Our Construction

We start by giving a high-level overview of the reusable MPC, which we call $\mathsf{r.MPC}$. Recall from Section 2.3 that round one of $\mathsf{r.MPC}$ essentially just consists of round one of an $\mathsf{FMS.MPC}$ instance computing the circuit N. We refer to this as the 0'th (instance of) MPC. Now fix a circuit C to be computed in round two, and its representative string $p := \langle \mathsf{C} \rangle$, which we'll take to be length m. This string p fixes a root-to-leaf path in a binary tree of MPCs that the parties will compute. In round two, the parties compute round two of the 0'th MPC, plus m (garbled circuit, encrypted labels) pairs. Each of these is used to compute an MPC in the output phase of $\mathsf{r.MPC}$. The first $m-1$ of these MPCs compute N, and the m'th MPC computes C.

In the first round of $\mathsf{r.MPC}$, each party i also chooses randomness r_i, which will serve as the root for a binary tree of random values generated as in [GGM84] by a PRG $(\mathsf{G}_0, \mathsf{G}_1)$. Below, we set $r_{i,0} := r_i$, where the 0 refers to the fact that the 0'th MPC will be computing the circuit N on input that includes $\{r_{i,0}\}_{i\in[n]}$. The string p then generates a sequence of values $r_{i,1}, \ldots, r_{i,m}$ by $r_{i,d} := \mathsf{G}_{p[d]}(r_{i,d-1})$. The d'th MPC will be computing the circuit N on input that includes $\{r_{i,d}\}_{i\in[n]}$.

Now, it remains to show how the m (garbled circuit, encrypted labels) pairs output by each party in round two can be used to reconstruct each of the m MPC outputs, culminating in C. We use a repeated application of the mechanism developed in the last section. In particular, the d'th garbled circuit output by party i computes their second round message of the d'th MPC. The input labels are encrypted using randomness derived from party i's root randomness r_i. Specifically, as in last section, we use a PRF to

compute a $2 \times \ell$ grid of keys, which will be used to LabEnc the $2 \times \ell$ grid of input labels. The key to this PRF will be generated by a PRG $(\mathsf{H}_0, \mathsf{H}_1)$ applied to $r_{i,d-1}$. Since we are branching based on the bit $p[d]$, the key will be set to $\mathsf{H}_{p[d]}(r_{i,d-1})$.

Likewise, the d'th MPC (for $d < m$), using inputs $\{r_{i,d}\}_{i\in[n]}$, computes two instances of the first round of the d+1'st MPC, the "left child" using inputs $\{\mathsf{G}_0(r_{i,d})\}_{i\in[n]}$ and the "right child" using inputs $\{\mathsf{G}_1(r_{i,d})\}_{i\in[n]}$. It then uses the PRF key $\mathsf{H}_0(r_{i,d})$ to output the ℓ keys corresponding to party i's left child first round message, and the key $\mathsf{H}_1(r_{i,d})$ to output the ℓ keys corresponding to party i's right child first round message.

Finally, in the output phase of $\mathsf{r.MPC}$, all parties can recover party i's second round message of the d'th MPC, by first using the output of the $d-1$'st MPC to decrypt party i's input labels corresponding to its first round message of the d'th MPC, and then using those labels to evaluate its d'th garbled circuit, finally recovering the second round message. Once all of the d'th second round messages have been recovered, the output may be reconstructed. Note that this output is exactly the set of keys necessary to repeat the process for the $d+1$'st MPC. Eventually, the parties will arrive at the m'th MPC, which allows them to recover the final output $\mathsf{C}(x_1, \ldots, x_n)$. One final technicality is that each party's second round message for each MPC may be generated along with a secret state. We cannot leak this state to other parties in the output phase, so in the second round of $\mathsf{r.MPC}$, parties will actually garble circuits that compute their second round (state, message) pair, encrypt the state with their own secret key, and then output the encrypted state plus the message in the clear. In the output phase, each party i can decrypt their own state, (but not anyone else's) and use their state to reconstruct the output of each MPC.

We present the formal construction in Figure 8. It is given for functionalities C where every party receives the same output, which is without loss of generality. Note that the circuit N used by the construction is defined immediately after in Figure 9.

A Reusable MPC potocol $(\mathsf{r.MPC}_1, \mathsf{r.MPC}_2, \mathsf{r.MPC}_3)$

Ingredients:

- A two-round first message succinct MPC protocol $(\mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$.
- A garbled circuit scheme $(\mathsf{Garble}, \mathsf{GEval})$.
- A label encryption scheme $(\mathsf{LabEnc}, \mathsf{LabDec})$.
- A length-quadrupling $\mathsf{PRG} = (\mathsf{G}_0, \mathsf{G}_1, \mathsf{H}_0, \mathsf{H}_1)$.
- A private-key encryption scheme $(\mathsf{enc}, \mathsf{dec})$ and associated function Wrap.

$\mathsf{r.MPC}_1(1^\lambda, \mathsf{CRS}, i, x_i)$**:**

1. Draw $r_i \leftarrow \{0,1\}^\lambda$, and compute $(\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, r_i))$.
2. Sample a key $\mathsf{sk}_i \leftarrow \{0,1\}^\lambda$ for $(\mathsf{enc}, \mathsf{dec})$.
3. Output $(\mathsf{r.st}_i^{(1)}, \mathsf{r.msg}_i^{(1)})$ where $\mathsf{r.st}_i^{(1)} := (\mathsf{st}_i^{(1)}, r_i, \mathsf{sk}_i)$ and $\mathsf{r.msg}_i^{(1)} := \mathsf{msg}_i^{(1)}$.

$\mathsf{r.MPC}_2(\mathsf{C}, \mathsf{r.st}_i^{(1)} = (\mathsf{st}_i^{(1)}, r_i, \mathsf{sk}_i), \{\mathsf{msg}_j^{(1)}\}_{j\in[n]})$**:**

1. Set $p := \langle \mathsf{C} \rangle$, and $r_{i,0} := r_i$.
2. Compute $(\mathsf{st}_i^{(2)}, \mathsf{msg}_i^{(2)}) \leftarrow \mathsf{MPC}_2(\mathsf{N}, \mathsf{st}_i^{(1)}, \{\mathsf{msg}_j^{(1)}\}_{j\in[n]})$.
3. For each $d \in [m]$,
   (a) If $d < m$, then compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot, \cdot)))$, else, compute $(\tilde{\mathsf{C}}_{i,m}, \overline{\mathsf{lab}}_{i,m}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot, \cdot)))$.
   (b) Set $\overline{K}_{i,d} := \{K_{t,b}\}_{t\in[\ell], b\in\{0,1\}}$, where each $K_{t,b} := \mathsf{PRF}(\mathsf{H}_{p[d]}(r_{i,d-1}), (t,b))$.
   (c) Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.
   (d) Compute $r_{i,d} := \mathsf{G}_{p[d]}(r_{i,d-1})$.
4. Output $\mathsf{r.st}_i^{(2)} := (\mathsf{st}_i^{(2)}, \mathsf{sk}_i)$ and $\mathsf{r.msg}_i^{(2)} := \left(\mathsf{msg}_i^{(2)}, \{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right)$.

$\mathsf{r.MPC}_3\left(\mathsf{r.st}_i^{(2)} = (\mathsf{st}_i^{(2)}, \mathsf{sk}_i), \left\{\mathsf{r.msg}_j^{(2)} = \left(\mathsf{msg}_j^{(2)}, \{\tilde{\mathsf{C}}_{j,d}, \overline{\mathsf{elab}}_{j,d}\}_{d\in[m]}\right)\right\}_{j\in[n]}\right)$**:**

1. Compute $\begin{pmatrix} \widehat{K}^0_{1,1} \dots \widehat{K}^0_{n,1} \\ \widehat{K}^1_{1,1} \dots \widehat{K}^1_{n,1} \end{pmatrix} := \mathsf{MPC}_3(\mathsf{st}_i^{(2)}, \{\mathsf{msg}_j^{(2)}\}_{j\in[n]})$.
2. For each $d \in [m]$:
   (a) For each $j \in [n]$:
      i. Compute $\widehat{\mathsf{lab}} := \mathsf{LabDec}(\widehat{K}_{j,d}^{p[d]}, \overline{\mathsf{elab}}_{j,d})$.
      ii. Evaluate $(\mathsf{est}_j^{(2)}, \mathsf{msg}_j^{(2)}) := \mathsf{GEval}(\tilde{\mathsf{C}}_{j,d}, \widehat{\mathsf{lab}})$.
   (b) Decrypt $\mathsf{st}_i^{(2)} := \mathsf{dec}(\mathsf{sk}_i, \mathsf{est}_i^{(2)})$.
   (c) If $d < m$, compute $\begin{pmatrix} \widehat{K}^0_{1,d+1} \dots \widehat{K}^0_{n,d+1} \\ \widehat{K}^1_{1,d+1} \dots \widehat{K}^1_{n,d+1} \end{pmatrix} := \mathsf{MPC}_3(\mathsf{st}_i^{(2)}, \{\mathsf{msg}_j^{(2)}\}_{j\in[n]})$, else, compute $y := \mathsf{MPC}_3(\mathsf{st}_i^{(2)}, \{\mathsf{msg}_j^{(2)}\}_{j\in[n]})$.
3. Output y.

Figure 8: A Reusable MPC potocol $(\mathsf{r.MPC}_1, \mathsf{r.MPC}_2, \mathsf{r.MPC}_3)$

Circuit N

Input: $(x_1, r_1) \ldots, (x_n, r_n)$.

1. For each $i \in [n]$ (generating the left child MPC_1):
   (a) Compute $z_i^0 := (\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_0(r_i)))$.
   (b) For each $t \in [\ell]$, set $K_{i,t}^0 = \mathsf{PRF}(\mathsf{H}_0(r_i), (t, z_i^0[t]))$.
   (c) Set $\widehat{K}_i^0 = \{K_{i,t}^0\}_{t \in [\ell]}$.
2. For each $i \in [n]$ (generating the right child MPC_1):
   (a) Compute $z_i^1 := (\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_1(r_i)))$.
   (b) For each $t \in [\ell]$, set $K_{i,t}^1 = \mathsf{PRF}(\mathsf{H}_1(r_i), (t, z_i^1[t]))$.
   (c) Set $\widehat{K}_i^1 = \{K_{i,t}^1\}_{t \in [\ell]}$.

Output: $\begin{pmatrix} \widehat{K}_1^0 \ldots \widehat{K}_n^0 \\ \widehat{K}_1^1 \ldots \widehat{K}_n^1 \end{pmatrix}$.

Figure 9: The (randomized) circuit N.

Theorem 6.1. *Let $\mathcal{X} \in \{$semi-honest in the plain or CRS model, malicious in the CRS model$\}$. Assuming a first message succinct $\mathcal{X}$ two-round MPC protocol, there exists an $\mathcal{X}$ reusable two-round MPC protocol.*

6.2 The Simulator

In this section, we present a hybrid simulator, $\mathsf{r.Sim}_k^{\mathcal{A}}$, where for depth parameter k, the first k MPCs carried out in the second round of $\mathsf{r.MPC}$ are simulated, while the remaining are executed honestly. When k is the size of the largest circuit computed, this $\mathsf{r.Sim}_k^{\mathcal{A}}$ is the full simulator, which does not take take as input the inputs of honest parties.

Recall that any C to be computed fixes a string $p := \langle \mathsf{C} \rangle$. This string p in turn fixes a sequence of strings $r_{i,0}, r_{i,1}, \ldots, r_{i,m}$ for each party i, where $r_{i,0}$ is always set to the r_i chosen by party i in the first round. In the simulated execution up to depth k, corrupt party messages will be generated using this sequence of values $r_{i,0}, r_{i,1}, \ldots, r_{i,m}$. However, honest party messages up to depth k will be computed with uniform and independent randomness. Eventually, for each honest party i, the string $r_{i,k}$ will be chosen at random. Then the values $r_{i,k+1}, \ldots, r_{i,m}$ will be fixed following the protocol, by $r_{i,d} := \mathsf{G}_{p[d]}(r_{i,d-1})$.

Now, we explain each step of the simulator $\mathsf{r.Sim}_k^{\mathcal{A}}$, where $k \in [m]$. The simulator is given inputs $\{x_i\}_{i\in[n]}$ (if $k = m$, then it only needs the corrupt party inputs), and repeatedly uses the $\mathsf{FMS.MPC}$ simulator $\mathsf{FMS.Sim}$.

Steps 1-2. The simulator first generates input randomness r_i for the corrupt parties, and secret keys sk_i for the honest parties.

Steps 3-5. It begins to simulate the 0'th MPC, whose first message constitutes the first message of $\mathsf{r.MPC}$. In step 3, it makes use of the $\mathsf{FMS.MPC}$ simulator, calling this instance $\mathsf{FMS.MPC}_0$, to generate the first round messages on behalf of the honest parties. In step 4, it sends these to $\mathcal{A}$ as the first round messages of $\mathsf{r.MPC}$. When $\mathcal{A}$ responds with its first round messages, it extracts inputs $\{\tilde{x}_i, \tilde{r}_i\}_{i\in I}$. For each $i \in I$, it sets $r_{i,0}$ to the extracted $\tilde{r}_i$, which as explained earlier, serves as the root of party i's tree of randomness.

Step 6. The adversary chooses the circuit C to compute.

Step 7. At this point, the simulator must generate second round messages on behalf of the honest parties. One component of a $\mathsf{r.MPC}$ second round message is the second round message of the 0'th MPC. Step 7 will generate these messages $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$ on behalf of the honest parties. To do so, it uses the simulator $\mathsf{FMS.MPC}_0$ initialized in earlier steps. In order to use $\mathsf{FMS.MPC}_0$, it must answer $\mathsf{FMS.MPC}_0$'s call to its ideal functionality. Thus, steps (a) and (b) compute the output of circuit N, using randomness derived from $r_{i,0}$ for corrupt parties, and uniform independent randomness for honest parties. This output is given to $\mathsf{FMS.MPC}_0$, who responds with second round messages for the 0'th MPC.

Step 8. The other component of a $\mathsf{r.MPC}$ second round message is a sequence of (garbled circuit, encrypted labels) pairs for each party. These are computed in step 8, which is split into 3 sections: (a) the simulated levels before k, (b) the "transition" level k, and (c) the honestly generated levels after k.

In section (a), the d'th MPC is simulated, and its output is used to simulate the d'th garbled circuit for each honest party. Moreover, this d'th MPC computing the circuit N is done so using uniform and independent randomness for each of the honest parties. In more detail, step (i) generates the next randomness $r_{i,d}$ for each corrupt party. Steps $(ii) - (iv)$ are similar to $3 - 5$ above, where $\mathsf{FMS.Sim}$ is initialized as $\mathsf{FMS.Sim}_d$. Observe that the corrupt party messages $\{\mathsf{msg}_{i,d}^{(1)}\}_{i\in I}$ have been generated in a previous step, so there is no need to involve $\mathcal{A}$ here. Steps $(v) - (vii)$ are similar to step 7 above, where the circuit N is computed and its output given to $\mathsf{FMS.Sim}_d$, who responds with the simulated second round messages for honest parties. In step $(viii)$ these second round messages are used to simulate the garbled circuit for each honest party. The simulated labels are encrypted using keys drawn for each honest party at the previous

depth. To fill out the $2 \times \ell$ grid of encryptions, each simulated label is also copied and encrypted under a new uniformly drawn key.

Section (b) is very similar to section (a). The major difference is that in step (v), the circuit N is computed honestly with respect to the randomness $r_{i,d}$ chosen for the honest parties $i \in H$. Each of these $r_{i,d}$ becomes the root randomness of the remaining sub-tree for each honest party. The other difference is that this may be the final level, in which case $\mathsf{FMS.Sim}_d$ should actually be answered with the value $\mathsf{C}(x_1, \ldots, x_n)$. This (step (vi)) is where the $\mathsf{r.MPC}$ simulator uses its ideal functionality.

Section (c) generates the remaining (garbled circuit, encrypted labels) pairs as in the real execution of $\mathsf{r.MPC}$. The simulator exactly follows the step 3 of $\mathsf{r.MPC}_2$.

Steps 9-11. Now that the second round messages of $\mathsf{r.MPC}$ for honest parties have been simulated, they are sent to $\mathcal{A}$, who responds with the second round messages on behalf of the corrupt parties. The messages corresponding to the 0'th MPC are sent to $\mathsf{FMS.Sim}_0$. If they are well-formed, then execution continues, otherwise, the honest parties are instructed to output $\perp$. Then, in step 10, the messages are combined to produce the honest party outputs. For every level d up to k, observe that the keys necessary to decrypt the encrypted labels have already been generated. The garbled circuits are then evaluated, and the resulting second round messages for the d'th MPC are sent to $\mathsf{FMS.Sim}_d$. If they are well-formed, then execution continues, otherwise, the honest parties are instructed to output $\perp$. After level k, reconstruction of the honest party outputs proceeds as in $\mathsf{r.MPC}_3$.

Notation:

- For any $i \in [n]$, let $\mathsf{w} := |\mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)|$ and $h := |(\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)})|$, where $(\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, r_i))$. If this instance of (FMS) MPC is executed at the d-th level of the tree, and the bit c indicates left or right child, the message-state pair is denoted by $(\mathsf{st}_{i,d,c}^{(1)}, \mathsf{msg}_{i,d,c}^{(1)})$.
- Let $p^*[d]$ denote the prefix of string p of length $d-1$, where we set $p^*[1]$ to a special symbol ϵ. For each $i \in [n], d \in [m], p^*[d] \in \{0,1\}^{d-1}, c \in \{0,1\}$, we let $\mathcal{R}((i, p^*[d], c), \cdot)$ denote a uniformly random function from $\{0,1\}^* \to \{0,1\}^\lambda$, and $\mathcal{U}(i, p^*[d], c), \mathcal{V}(i, p^*[d], c)$ denote uniformly random strings of length λ.

Remark 6.2. *The following proof is stated only for the malicious case. The semi-honest case is very similar, the only difference being that we need to argue that the simulator never sends $\perp$ to its ideal functionality. It suffices to argue that each $\mathsf{msg}_{j,d}^{(2)}$ recovered in step 10.(b).(i).(B) of the simulation corresponds to the honestly generated second round message of the d'th MPC for party j. This is straight-forward to see from the robustness property of* LabEnc*, since decrypting any wrong label happens with negligible probability due to the use of* $\mathsf{rob.enc}$*.*

Intermediate Simulator for $k \in [m]$, $\mathsf{r.Sim}_k^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i \in [n]})$:

1. For each $i \in I$, sample $r_i \leftarrow \{0,1\}^\lambda$.
2. For each $i \in H$, sample $\mathsf{sk}_i \leftarrow \{0,1\}^\lambda$.
3. Initialize $\mathsf{FMS.Sim}$ with inputs $(1^\lambda, z, I, \{x_i, r_i\}_{i \in I})$. Denote this instance of the simulator by $\mathsf{FMS.Sim}_0$. Run $\mathsf{FMS.Sim}_0$ to obtain $\left(\mathsf{CRS}, \{\mathsf{msg}_{i,0}^{(1)}\}_{i \in H}\right)$.
4. Send $\left(\mathsf{CRS}, \{\mathsf{msg}_{i,0}^{(1)}\}_{i \in H}\right)$ to $\mathcal{A}$ on behalf of the honest parties. $\mathcal{A}$ responds with $\{\mathsf{msg}_{i,0}^{(1)}\}_{i \in I}$. Forward $\{\mathsf{msg}_{i,0}^{(1)}\}_{i \in I}$ to $\mathsf{FMS.Sim}_0$.
5. $\mathsf{FMS.Sim}_0$ queries $\{\tilde{x}_i, \tilde{r}_i\}_{i \in I}$ to its ideal functionality $\mathcal{F}_{\mathsf{N}}$. For each $i \in I$, set $x_i := \tilde{x}_i$ and $r_{i,0} := \tilde{r}_i$.
6. $\mathcal{A}$ outputs the description of a circuit C_q, set $p_q := \langle \mathsf{C}_q \rangle$. The following steps are repeated for each circuit query $q \in [Q]$.

7. Compute $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$ as follows.
   (a) For each $i \in I, c \in \{0,1\}$,
      i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,0})))$.
      ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathsf{PRF}(\mathsf{H}_c(r_{i,0}), (t, z_{i,1}^c[t]))$.
      iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.
   (b) For each $i \in H, c \in \{0,1\}$,
      i. Skip.
      ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathcal{R}((i, p_q^*[1], c), (t, 0))$.
      iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.
   (c) Provide $\begin{pmatrix} \widehat{K}_{1,1}^0 \dots \widehat{K}_{n,1}^0 \\ \widehat{K}_{1,1}^1 \dots \widehat{K}_{n,1}^1 \end{pmatrix}$ to $\mathsf{FMS.Sim}_0$, who responds with $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$.

8. Compute $\left\{\{\widetilde{\mathsf{C}}_{i,d}, \overline{\widetilde{\mathsf{lab}}}_{i,d}\}_{d\in[m]}\right\}_{i\in H}$ as follows.
   (a) For $d < k$,
      i. For each $i \in I$, compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$.
      ii. Initialize $\mathsf{FMS.Sim}$ with inputs $(1^\lambda, z, I, \{x_i, r_{i,d}\}_{i\in I})$. Denote this instance of the simulator by $\mathsf{FMS.Sim}_d$. Run $\mathsf{FMS.Sim}_d$ to obtain $\left(\mathsf{CRS}, \{\mathsf{msg}_{i,d}^{(1)}\}_{i\in H}\right)$.
      iii. Ignore $\mathsf{FMS.Sim}_d$'s output, and send back $\{\mathsf{msg}_{i,d,p_q[d]}^{(1)}\}_{i\in I}$.
      iv. $\mathsf{FMS.Sim}_d$ queries $\{\widetilde{x}_i, \widetilde{r}_{i,d}\}_{i\in I}$ to its ideal functionality, which is ignored.
      v. For each $i \in I, c \in \{0,1\}$,
         A. Compute $z_{i,d+1}^c := (\mathsf{st}_{i,d+1,c}^{(1)}, \mathsf{msg}_{i,d+1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,d})))$.
         B. For each $t \in [\ell]$, set $K_{i,t}^c := \mathsf{PRF}(\mathsf{H}_c(r_{i,d}), (t, z_{i,d+1}^c[t]))$.
         C. Set $\widehat{K}_{i,d+1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.
      vi. For each $i \in H, c \in \{0,1\}$,
         A. For each $t \in [\ell]$, set $K_{i,t}^c := \mathcal{R}((i, p_q^*[d+1], c), (t, 0))$.
         B. Set $\widehat{K}_{i,d+1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.
      vii. Provide $\begin{pmatrix} \widehat{K}_{1,d+1}^0 \dots \widehat{K}_{n,d+1}^0 \\ \widehat{K}_{1,d+1}^1 \dots \widehat{K}_{n,d+1}^1 \end{pmatrix}$ to $\mathsf{FMS.Sim}_d$, who responds with $\{\mathsf{msg}_{i,d}^{(2)}\}_{i\in H}$.
      viii. For each $i \in H$,
         A. Set $\mathsf{est}_{i,d}^{(2)} := \mathsf{enc}(\mathsf{sk}_i, \vec{0})$ and compute $(\widetilde{\mathsf{C}}_{i,d}, \widehat{\overline{\mathsf{lab}}}_{i,d}) \leftarrow \mathsf{GSim}(1^\mathsf{w}, 1^h, (\mathsf{est}_{i,d}^{(2)}, \mathsf{msg}_{i,d}^{(2)}))$.
         B. Set $\overline{\widetilde{\mathsf{lab}}}_{i,d} := (\widehat{\mathsf{lab}}_{i,d}, \widehat{\mathsf{lab}}_{i,d})$.
         C. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$ where each $K_{i,t,b} := \mathcal{R}((i, p_q^*[d], p_q[d]), (t, b))$.
         D. Compute $\overline{\widetilde{\mathsf{elab}}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\widetilde{\mathsf{lab}}}_{i,d})$.
   (b) For $d = k$,
      i. For each $i \in I$, compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$, and for each $i \in H$, set $r_{i,d} := \mathcal{U}(i, p_q^*[d], p_q[d])$.
      ii. Initialize $\mathsf{FMS.Sim}$ with inputs $(1^\lambda, z, I, \{x_i, r_{i,d}\}_{i\in I})$. Denote this instance of the simulator by $\mathsf{FMS.Sim}_d$. Run $\mathsf{FMS.Sim}_d$ to obtain $\left(\mathsf{CRS}, \{\mathsf{msg}_{i,d}^{(1)}\}_{i\in H}\right)$.
      iii. Ignore $\mathsf{FMS.Sim}_d$'s output, and send back $\{\mathsf{msg}_{i,d,p_q[d]}^{(1)}\}_{i\in I}$.
      iv. $\mathsf{FMS.Sim}_d$ queries $\{\widetilde{x}_i, \widetilde{r}_{i,d}\}_{i\in I}$ to its ideal functionality, which is ignored.

v. If $k \neq m$,
   A. For each $i \in [n], c \in \{0,1\}$,
      - Compute $z^c_{i,d+1} := (\mathsf{st}^{(1)}_{i,d+1,c}, \mathsf{msg}^{(1)}_{i,d+1,c}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,d})))$.
      - For each $t \in [\ell]$, set $K^c_{i,t} := \mathsf{PRF}(\mathsf{H}_c(r_{i,d}), (t, z^c_{i,d+1}[t]))$.
      - Set $\widehat{K}^c_{i,d+1} := \{K^c_{i,t}\}_{t\in[\ell]}$.
   B. Provide $\begin{pmatrix} \widehat{K}^0_{1,d+1} \dots \widehat{K}^0_{n,d+1} \\ \widehat{K}^1_{1,d+1} \dots \widehat{K}^1_{n,d+1} \end{pmatrix}$ to $\mathsf{FMS.Sim}_d$, who responds with $\{\mathsf{msg}^{(2)}_{i,d}\}_{i\in H}$.

vi. Otherwise, if $k = m$,
   A. Query the ideal functionality $\mathcal{F}_\mathsf{C}$ with $\{x_i\}_{i\in I}$ and receive $\{y_i\}_{i\in I}$.
   B. Provide $\{y_i\}_{i\in I}$ to $\mathsf{FMS.Sim}_d$, who responds with $\{\mathsf{msg}^{(2)}_{i,d}\}_{i\in H}$.

vii. For each $i \in H$,
   A. Set $\mathsf{est}^{(2)}_{i,d} := \mathsf{enc}(\mathsf{sk}_i, \vec{0})$ and compute $(\widetilde{\mathsf{C}}_{i,d}, \widehat{\overline{\mathsf{lab}}}_{i,d}) \leftarrow \mathsf{GSim}(1^\mathsf{w}, 1^h, (\mathsf{est}^{(2)}_{i,d}, \mathsf{msg}^{(2)}_{i,d}))$.
   B. Set $\overline{\overline{\mathsf{lab}}}_{i,d} := (\widehat{\mathsf{lab}}_{i,d}, \widehat{\overline{\mathsf{lab}}}_{i,d})$.
   C. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$ where each $K_{i,t,b} := \mathcal{R}((i, p^*_q[d], p_q[d]), (t,b))$.
   D. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

(c) For $k < d \leq m$, and $i \in H$,
   i. If $d < m$, then compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$,
   else, compute $(\tilde{\mathsf{C}}_{i,m}, \overline{\mathsf{lab}}_{i,m}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)))$.
   ii. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathsf{PRF}(\mathsf{H}_{p_q[d]}(r_{i,d-1}), (t,b))$.
   iii. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.
   iv. Compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$.

9. Send $\left\{\left(\mathsf{msg}^{(2)}_{i,0}, \{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right)\right\}_{i\in H}$ to $\mathcal{A}$ on behalf of the honest parties. $\mathcal{A}$ responds with $\left\{\left(\mathsf{msg}^{(2)}_{i,0}, \{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right)\right\}_{i\in I}$.

10. Compute $\{y_i\}_{i\in H}$ as follows.

(a) Provide the $\{\mathsf{msg}^{(2)}_{i,0}\}_{i\in I}$ to $\mathsf{FMS.Sim}_0$. If $\mathsf{FMS.Sim}_0$ outputs $\mathsf{continue}$ then continue and otherwise set $\{y_i\}_{i\in H} := \perp$ and skip to step 11.

(b) For $d \leq k$,
   i. For each $j \in [n]$,
      A. Compute $\widehat{\mathsf{lab}}_{j,d} := \mathsf{LabDec}(\widehat{K}^{p_q[d]}_{j,d}, \overline{\mathsf{elab}}_{j,d})$.
      B. Evaluate $(\mathsf{est}^{(2)}_{j,d}, \mathsf{msg}^{(2)}_{j,d}) := \mathsf{GEval}(\tilde{\mathsf{C}}_{j,d}, \widehat{\mathsf{lab}}_{j,d})$.
   ii. Provide the $\{\mathsf{msg}^{(2)}_{i,d}\}_{i\in I}$ to $\mathsf{FMS.Sim}_d$. If $\mathsf{FMS.Sim}_d$ outputs $\mathsf{continue}$ then continue and otherwise set $\{y_i\}_{i\in H} := \perp$ and skip to step 11.

(c) For $k < d \leq m$, and $i \in H$,
   i. For each $j \in [n]$,
      A. Compute $\widehat{\mathsf{lab}}_{j,d} := \mathsf{LabDec}(\widehat{K}^{p_q[d]}_{j,d}, \overline{\mathsf{elab}}_{j,d})$.
      B. Evaluate $(\mathsf{est}^{(2)}_{j,d}, \mathsf{msg}^{(2)}_{j,d}) := \mathsf{GEval}(\tilde{\mathsf{C}}_{j,d}, \widehat{\mathsf{lab}}_{j,d})$.
   ii. Decrypt $\mathsf{st}^{(2)}_{i,d} := \mathsf{dec}(\mathsf{sk}_i, \mathsf{est}^{(2)}_{i,d})$.
   iii. if $d < m$, compute $\begin{pmatrix} \widehat{K}^0_{1,d+1} \dots \widehat{K}^0_{n,d+1} \\ \widehat{K}^1_{1,d+1} \dots \widehat{K}^1_{n,d+1} \end{pmatrix} := \mathsf{MPC}_3(\mathsf{st}^{(2)}_{i,d}, \{\mathsf{msg}^{(2)}_{j,d}\}_{j\in[n]})$,
   else, compute $y_i := \mathsf{MPC}_3(\mathsf{st}^{(2)}_{i,m}, \{\mathsf{msg}^{(2)}_{j,m}\}_{j\in[n]})$.

11. Output $\{y_i\}_{i\in H}$ and $\mathcal{A}$'s output. If $q \neq Q$ go back to step 6.

6.3 Hybrids

We prove the security of $\mathsf{r.MPC}$ in two sequences of hybrids. The first sequence of hybrids begins with real world execution $\mathsf{Hyb}_0^{\mathcal{A}}$, and ends with $\mathsf{r.Sim}_1^{\mathcal{A}}$. The second sequence begins with $\mathsf{r.Sim}_k^{\mathcal{A}}$ and end with $\mathsf{r.Sim}_{k+1}^{\mathcal{A}}$, for any $k \in [m-1]$. Noting that $\mathsf{r.Sim}_m^{\mathcal{A}}$ does not require the inputs of honest parties completes the proof. The first sequence is given below.

$\underline{\mathsf{Hyb}_0^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ Hybrid $\mathsf{Hyb}_0^{\mathcal{A}}$ corresponds to the real world execution of $\mathsf{r.MPC}$ in the presence of the adversary $\mathcal{A}$.

1. Skip.
2. For each $i \in H$, sample $r_{i,0}, \mathsf{sk}_i \leftarrow \{0,1\}^\lambda$.
3. Sample CRS from the prescribed distribution. Then, for each $i \in H$, compute $(\mathsf{st}_{i,0}^{(1)}, \mathsf{msg}_{i,0}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, r_{i,0}))$.
4. Send $\left(\mathsf{CRS}, \{\mathsf{msg}_{i,0}^{(1)}\}_{i\in H}\right)$ to $\mathcal{A}$ on behalf of the honest parties. $\mathcal{A}$ responds with $\{\mathsf{msg}_{i,0}^{(1)}\}_{i\in I}$.
5. Skip.
6. $\mathcal{A}$ outputs the description of a circuit C_q, set $p_q := \langle \mathsf{C}_q \rangle$. The following steps are repeated for each circuit query $q \in [Q]$.
7. Compute $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$ as follows.
   (a) Skip.
   (b) Skip.
   (c) For each $i \in H$, compute $(\mathsf{st}_{i,0}^{(2)}, \mathsf{msg}_{i,0}^{(2)}) \leftarrow \mathsf{MPC}_2(\mathsf{N}, \mathsf{st}_{i,0}^{(1)}, \{\mathsf{msg}_{j,0}^{(1)}\}_{j\in[n]})$.
8. Compute $\left\{\{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right\}_{i\in H}$ as follows.
   (a) Skip.
   (b) Skip.
   (c) For $0 < d \le m$, and $i \in H$,
      i. If $d < m$, then compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$,
      else, compute $(\tilde{\mathsf{C}}_{i,m}, \overline{\mathsf{lab}}_{i,m}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)))$,
      ii. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell], b\in\{0,1\}}$, where each $K_{i,t,b} := \mathsf{PRF}(\mathsf{H}_{p_q[d]}(r_{i,d-1}), (t,b))$.
      iii. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.
      iv. Compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$.
9. Send $\left\{\left(\mathsf{msg}_{i,0}^{(2)}, \{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right)\right\}_{i\in H}$ to $\mathcal{A}$ on behalf of the honest parties. $\mathcal{A}$ responds with $\left\{\left(\mathsf{msg}_{i,0}^{(2)}, \{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right)\right\}_{i\in I}$.
10. Compute $\{y_i\}_{i\in H}$ as follows.
    (a) Compute $\begin{pmatrix} \widehat{K}_{1,1}^0 \dots \widehat{K}_{n,1}^0 \\ \widehat{K}_{1,1}^1 \dots \widehat{K}_{n,1}^1 \end{pmatrix} := \mathsf{MPC}_3(\mathsf{st}_{i,0}^{(2)}, \{\mathsf{msg}_{j,0}^{(2)}\}_{j\in[n]})$.
    (b) Skip.
    (c) For $0 < d \le m$, and $i \in H$,

i. For each $j \in [n]$,
   A. Compute $\widehat{\mathsf{lab}}_{j,d} := \mathsf{LabDec}(\widehat{K}_{j,d}^{p_q[d]}, \overline{\mathsf{elab}}_{j,d})$.
   B. Evaluate $(\mathsf{est}_{j,d}^{(2)}, \mathsf{msg}_{j,d}^{(2)}) := \mathsf{GEval}(\tilde{\mathsf{C}}_{j,d}, \widehat{\mathsf{lab}}_{j,d}))$.

ii. Decrypt $\mathsf{st}_{i,d}^{(2)} := \mathsf{dec}_{\mathsf{sk}_i}(\mathsf{est}_{i,d}^{(2)})$.

iii. if $d < m$, compute $\begin{pmatrix} \widehat{K}_{1,d}^0 \dots \widehat{K}_{n,d}^0 \\ \widehat{K}_{1,d}^1 \dots \widehat{K}_{n,d}^1 \end{pmatrix} := \mathsf{MPC}_3(\mathsf{st}_{i,d}^{(2)}, \{\mathsf{msg}_{j,d}^{(2)}\}_{j\in[n]})$,
else, compute $y_i := \mathsf{MPC}_3(\mathsf{st}_{i,m}^{(2)}, \{\mathsf{msg}_{j,m}^{(2)}\}_{j\in[n]})$.

11. Output $\{y_i\}_{i\in H}$ and $\mathcal{A}$'s output. If $q \neq Q$ go back to step 6.

$\mathsf{Hyb}_1^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]})$: In Hybrid $\mathsf{Hyb}_1^{\mathcal{A}}$, we simulate the 0'th MPC. Indistinguishability follows from the security of $\mathsf{FMS.MPC}$. Note that steps 7.(a) and 7.(b) are exactly the same, so we didn't necessarily have to branch based on I and H in this hybrid. This is for convenience, to clearly see the difference with the next hybrid.

1. For each $i \in I$ sample $r_i \leftarrow \{0,1\}^\lambda$.
2. For each $i \in H$, sample $r_{i,0}, \mathsf{sk}_i \leftarrow \{0,1\}^\lambda$.
3. Initialize $\mathsf{FMS.Sim}$ with inputs $(1^\lambda, z, I, \{x_i, r_i\}_{i\in I})$. Denote this instance of the simulator by $\mathsf{FMS.Sim}_0$. Run $\mathsf{FMS.Sim}_0$ to obtain $\left(\mathsf{CRS}, \{\mathsf{msg}_{i,0}^{(1)}\}_{i\in H}\right)$.
4. Send $\left(\mathsf{CRS}, \{\mathsf{msg}_{i,0}^{(1)}\}_{i\in H}\right)$ to $\mathcal{A}$ on behalf of the honest parties. $\mathcal{A}$ responds with $\{\mathsf{msg}_{i,0}^{(1)}\}_{i\in I}$.
5. $\mathsf{FMS.Sim}_0$ queries $\{\tilde{x}_i, \tilde{r}_i\}_{i\in I}$ to its ideal functionality $\mathcal{F}_{\mathsf{N}}$. For each $i \in I$, set $x_i := \tilde{x}_i$ and $r_{i,0} := \tilde{r}_i$.
6. $\mathcal{A}$ outputs the description of a circuit C_q, set $p_q := \langle \mathsf{C}_q \rangle$. The following steps are repeated for each circuit query $q \in [Q]$.
7. Compute $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$ as follows.
   (a) For each $i \in I, c \in \{0,1\}$,
      i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,0})))$.
      ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathsf{PRF}(\mathsf{H}_c(r_{i,0}), (t, z_{i,1}^c[t]))$.
      iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.
   (b) For each $i \in H, c \in \{0,1\}$,
      i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,0})))$.
      ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathsf{PRF}(\mathsf{H}_c(r_{i,0}), (t, z_{i,1}^c[t]))$.
      iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.
   (c) Provide $\begin{pmatrix} \widehat{K}_{1,1}^0 \dots \widehat{K}_{n,1}^0 \\ \widehat{K}_{1,1}^1 \dots \widehat{K}_{n,1}^1 \end{pmatrix}$ to $\mathsf{FMS.Sim}_0$, who responds with $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$.
8. Compute $\left\{\{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right\}_{i\in H}$ as follows.
   (a) Skip.
   (b) Skip.
   (c) For $0 < d \leq m$, and $i \in H$,

i. If $d < m$, then compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$,
else, compute $(\tilde{\mathsf{C}}_{i,m}, \overline{\mathsf{lab}}_{i,m}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)))$.

ii. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathsf{PRF}(\mathsf{H}_{p_q[d]}(r_{i,d-1}), (t,b))$.

iii. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

iv. Compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$.

9. Send $\left\{\left(\mathsf{msg}^{(2)}_{i,0}, \{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right)\right\}_{i\in H}$ to $\mathcal{A}$ on behalf of the honest parties. $\mathcal{A}$ responds with $\left\{\left(\mathsf{msg}^{(2)}_{i,0}, \{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right)\right\}_{i\in I}$.

10. Compute $\{y_i\}_{i\in H}$ as follows.

(a) Provide the $\{\mathsf{msg}^{(2)}_{i,0}\}_{i\in I}$ to $\mathsf{FMS.Sim}_0$. If $\mathsf{FMS.Sim}_0$ outputs continue then continue and otherwise set $\{y_i\}_{i\in H} := \perp$ and skip the next step.

(b) Skip.

(c) For $0 < d \le m$, and $i \in H$,

i. For each $j \in [n]$,

A. Compute $\widehat{\mathsf{lab}}_{j,d} := \mathsf{LabDec}(\widehat{K}^{p_q[d]}_{j,d}, \overline{\mathsf{elab}}_{j,d})$.

B. Evaluate $(\mathsf{est}^{(2)}_{j,d}, \mathsf{msg}^{(2)}_{j,d}) := \mathsf{GEval}(\tilde{\mathsf{C}}_{j,d}, \widehat{\mathsf{lab}}_{j,d}))$.

ii. Decrypt $\mathsf{st}^{(2)}_{i,d} := \mathsf{dec}(\mathsf{sk}_i, \mathsf{est}^{(2)}_{i,d})$.

iii. if $d < m$, compute $\begin{pmatrix}\widehat{K}^0_{1,d} \dots \widehat{K}^0_{n,d}\\ \widehat{K}^1_{1,d} \dots \widehat{K}^1_{n,d}\end{pmatrix} := \mathsf{MPC}_3(\mathsf{st}^{(2)}_{i,d}, \{\mathsf{msg}^{(2)}_{j,d}\}_{j\in[n]})$,
else, compute $y_i := \mathsf{MPC}_3(\mathsf{st}^{(2)}_{i,d}, \{\mathsf{msg}^{(2)}_{j,d}\}_{j\in[n]})$.

11. Output $\{y_i\}_{i\in H}$ and $\mathcal{A}$'s output. If $q \neq Q$ go back to step 6.

$\underline{\mathsf{Hyb}^{\mathcal{A}}_2(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}^{\mathcal{A}}_2$ we invoke the security of PRG at depth $d = 1$ for each honest party $i \in H$. We replace all honest parties PRG outputs for depth 1 with $\mathcal{U}(i, p^*_q[1], c)$ and $\mathcal{V}(i, p^*_q[1], c)$, for all circuit queries $q \in [Q]$ (note these values are the same for each query since they correspond to the single root of the tree).

7. Compute $\{\mathsf{msg}^{(2)}_{i,0}\}_{i\in H}$ as follows.

(a) For each $i \in I, c \in \{0,1\}$,

i. Compute $z^c_{i,1} := (\mathsf{st}^{(1)}_{i,1,c}, \mathsf{msg}^{(1)}_{i,1,c}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,0})))$.

ii. For each $t \in [\ell]$, set $K^c_{i,t} := \mathsf{PRF}(\mathsf{H}_c(r_{i,0}), (t, z^c_{i,1}[t]))$.

iii. Set $\widehat{K}^c_{i,1} := \{K^c_{i,t}\}_{t\in[\ell]}$.

(b) For each $i \in H, c \in \{0,1\}$,

i. Compute $z^c_{i,1} := (\mathsf{st}^{(1)}_{i,1,c}, \mathsf{msg}^{(1)}_{i,1,c}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathcal{U}(i, p^*_q[1], c)))$.

ii. For each $t \in [\ell]$, set $K^c_{i,t} := \mathsf{PRF}(\mathcal{V}(i, p^*_q[1], c), (t, z^c_{i,1}[t]))$.

iii. Set $\widehat{K}^c_{i,1} := \{K^c_{i,t}\}_{t\in[\ell]}$.

(c) Provide $\begin{pmatrix}\widehat{K}^0_{1,1} \dots \widehat{K}^0_{n,1}\\ \widehat{K}^1_{1,1} \dots \widehat{K}^1_{n,1}\end{pmatrix}$ to $\mathsf{FMS.Sim}_0$, who responds with $\{\mathsf{msg}^{(2)}_{i,0}\}_{i\in H}$.

8. Compute $\left\{\{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right\}_{i\in H}$ as follows.

(a) Skip.

(b) For $d = 1$,

i. For each $i \in H$, set $r_{i,d} := \mathcal{U}(i, p_q^*[d], p_q[d])$.

ii. Skip.

iii. Skip.

iv. Skip.

v. Skip.

vi. Skip.

vii. For each $i \in H$,

A. Compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$.

B. Skip.

C. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathsf{PRF}(\mathcal{V}(i, p_q^*[d], p_q[d]), (t, b))$.

D. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

(c) For $1 < d \le m$, and $i \in H$,

i. If $d < m$, then compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$, else, compute $(\tilde{\mathsf{C}}_{i,m}, \overline{\mathsf{lab}}_{i,m}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)))$.

ii. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathsf{PRF}(\mathsf{H}_{p_q[d]}(r_{i,d-1}), (t, b))$.

iii. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

iv. Compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$.

$\underline{\mathsf{Hyb}_3^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}_3^{\mathcal{A}}$, we invoke the security of PRF at depth $d = 1$ for each honest party $i \in H$.

7. Compute $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$ as follows.

(a) For each $i \in I, c \in \{0, 1\}$,

i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,0})))$.

ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathsf{PRF}(\mathsf{H}_c(r_{i,0}), (t, z_{i,1}^c[t]))$.

iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.

(b) For each $i \in H, c \in \{0, 1\}$,

i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathcal{U}(i, p_q^*[1], c)))$.

ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathcal{R}((i, p_q^*[1], c), (t, z_{i,1}^c[t]))$.

iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.

(c) Provide $\begin{pmatrix} \widehat{K}_{1,1}^0 \dots \widehat{K}_{n,1}^0 \\ \widehat{K}_{1,1}^1 \dots \widehat{K}_{n,1}^1 \end{pmatrix}$ to $\mathsf{FMS.Sim}_0$, who responds with $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$.

8. Compute $\left\{\{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right\}_{i\in H}$ as follows.

(a) Skip.

(b) For $d = 1$,

i. For each $i \in H$, set $r_{i,d} := \mathcal{U}(i, p_q^*[d], p_q[d])$.

ii. Skip.

iii. Skip.

iv. Skip.

v. Skip.

vi. Skip.

vii. For each $i \in H$,

A. Compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$.

B. Skip.

C. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathcal{R}((i, p_q^*[d], p_q[d]), (t, b))$.

D. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

(c) For $1 < d \le m$, and $i \in H$,

i. If $d < m$, then compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$, else, compute $(\tilde{\mathsf{C}}_{i,m}, \overline{\mathsf{lab}}_{i,m}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)))$.

ii. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathsf{PRF}(\mathsf{H}_{p_q[d]}(r_{i,d-1}), (t, b))$.

iii. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

iv. Compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$.

$\underline{\mathsf{Hyb}_4^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}_4^{\mathcal{A}}$ we copy the labels corresponding to the input that will be uncovered at depth 1 for all honest parties $i \in H$ and all circuit queries $q \in [Q]$. Indistinguishability follows from semantic security of LabEnc.

During the security reduction, we only rely on indistinguishability of ciphertexts encrypted under keys $\overline{K}_{i,d} \setminus \widehat{K}_{i,d}^c$, which are not used for anything beyond encypting the labels. Further, there is one challenge ciphertext for each party, label index $t \in [\ell]$ and circuit query, i.e. $|H|\ell Q$ in total.

7. Compute $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$ as follows.

(a) For each $i \in I, c \in \{0, 1\}$,

i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,0})))$.

ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathsf{PRF}(\mathsf{H}_c(r_{i,0}), (t, z_{i,1}^c[t]))$.

iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.

(b) For each $i \in H, c \in \{0, 1\}$,

i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathcal{U}(i, p_q^*[1], c)))$.

ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathcal{R}((i, p_q^*[1], c), (t, z_{i,1}^c[t]))$.

iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.

(c) Provide $\begin{pmatrix} \widehat{K}_{1,1}^0 \dots \widehat{K}_{n,1}^0 \\ \widehat{K}_{1,1}^1 \dots \widehat{K}_{n,1}^1 \end{pmatrix}$ to $\mathsf{FMS.Sim}_0$, who responds with $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$.

8. Compute $\left\{\{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right\}_{i\in H}$ as follows.

(a) Skip.

(b) For $d = 1$,

i. For each $i \in H$, set $r_{i,d} := \mathcal{U}(i, p_q^*[d], p_q[d])$.

ii. Skip.

iii. Skip.

iv. Skip.

v. Skip.

vi. Skip.

vii. For each $i \in H$,

A. Compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$.

B. Write $\overline{\mathsf{lab}}_{i,d}$ as $\{\mathsf{lab}_{i,d,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, and set $\overline{\mathsf{lab}}_{i,d} := (\widehat{\mathsf{lab}}_{i,d}, \widehat{\mathsf{lab}}_{i,d})$, where $\widehat{\mathsf{lab}}_{i,d} := \left\{\mathsf{lab}_{i,d,t,\left(z_{i,d}^{p_q[d]}\right)_t}\right\}_{t\in[\ell]}$.

C. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathcal{R}((i, p_q^*[d], p_q[d]), (t, b))$.

D. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

(c) For $1 < d \le m$, and $i \in H$,

i. If $d < m$, then compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$, else, compute $(\tilde{\mathsf{C}}_{i,m}, \overline{\mathsf{lab}}_{i,m}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)))$.

ii. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathsf{PRF}(\mathsf{H}_{p_q[d]}(r_{i,d-1}), (t, b))$.

iii. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

iv. Compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$.

$\underline{\mathsf{Hyb}_5^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}_5^{\mathcal{A}}$, we make the keys $K_{i,t}^c$ at depth $d = 1$ for all honest parties $i \in H$ independent of $z_{i,1}^c[t]$. Therefore, they are also independent of $(\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)})$ of MPC_1. The fact that each pair of messages are identical and LabEnc randomly permutes the output order of each ciphertext pair implies perfect indistinguishability.

7. Compute $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$ as follows.

(a) For each $i \in I, c \in \{0, 1\}$,

i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,0})))$.

ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathsf{PRF}(\mathsf{H}_c(r_{i,0}), (t, z_{i,1}^c[t]))$.

iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.

(b) For each $i \in H, c \in \{0, 1\}$,

i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathcal{U}(i, p_q^*[1], c)))$.

ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathcal{R}((i, p_q^*[1], c), (t, 0))$.

iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.

(c) Provide $\begin{pmatrix} \widehat{K}_{1,1}^0 \dots \widehat{K}_{n,1}^0 \\ \widehat{K}_{1,1}^1 \dots \widehat{K}_{n,1}^1 \end{pmatrix}$ to $\mathsf{FMS.Sim}_0$, who responds with $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$.

8. Compute $\left\{\{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right\}_{i\in H}$ as follows.

(a) Skip.

(b) For $d = 1$,

i. For each $i \in H$, set $r_{i,d} := \mathcal{U}(i, p_q^*[d], p_q[d])$.

ii. Skip.

iii. Skip.

iv. Skip.

v. Skip.

vi. Skip.

vii. For each $i \in H$,

A. Compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$.

B. Write $\overline{\mathsf{lab}}_{i,d}$ as $\{\mathsf{lab}_{i,d,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, and set $\overline{\mathsf{lab}}_{i,d} := (\widehat{\mathsf{lab}}_{i,d}, \widehat{\mathsf{lab}}_{i,d})$, where $\widehat{\mathsf{lab}}_{i,d} := \left\{\mathsf{lab}_{i,d,t,\left(z_{i,d}^{p_q[d]}\right)_t}\right\}_{t\in[\ell]}$.

C. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathcal{R}((i, p_q^*[d], p_q[d]), (t, b))$.

D. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

(c) For $1 < d \le m$, and $i \in H$,

i. If $d < m$, then compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$, else, compute $(\tilde{\mathsf{C}}_{i,m}, \overline{\mathsf{lab}}_{i,m}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)))$.

ii. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathsf{PRF}(\mathsf{H}_{p_q[d]}(r_{i,d-1}), (t, b))$.

iii. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

iv. Compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$.

$\underline{\mathsf{Hyb}_6^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}_6^{\mathcal{A}}$ we simulate the honest party's garbled circuits at depth $d = 1$ for each circuit query $q \in [Q]$. In total, we simulate $|H|Q$ garbled circuits. Indistinguishability follows from simulation security of garbling.

7. Compute $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$ as follows.

(a) For each $i \in I, c \in \{0, 1\}$,

i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,0})))$.

ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathsf{PRF}(\mathsf{H}_c(r_{i,0}), (t, z_{i,1}^c[t]))$.

iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.

(b) For each $i \in H, c \in \{0, 1\}$,

i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathcal{U}(i, p_q^*[1], c)))$.

ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathcal{R}((i, p_q^*[1], c), (t, 0))$.

iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.

(c) Provide $\begin{pmatrix} \widehat{K}_{1,1}^0 \dots \widehat{K}_{n,1}^0 \\ \widehat{K}_{1,1}^1 \dots \widehat{K}_{n,1}^1 \end{pmatrix}$ to $\mathsf{FMS.Sim}_0$, who responds with $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$.

8. Compute $\left\{\{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right\}_{i\in H}$ as follows.

(a) Skip.

(b) For $d = 1$,

i. For each $i \in H$, set $r_{i,d} := \mathcal{U}(i, p_q^*[d], p_q[d])$.

ii. Skip.

iii. Skip.

iv. Skip.

v. Set $(\mathsf{st}_{i,d}^{(2)}, \mathsf{msg}_{i,d}^{(2)}) \leftarrow \mathsf{MPC}_2(1^\lambda, \mathsf{CRS}, i, \mathsf{st}_{i,d,p_q[d]}^{(1)}, \{\mathsf{msg}_{j,d,p_q[d]}^{(1)}\}_{j\in[n]})$.

vi. Skip.

vii. For each $i \in H$,

A. Compute $(\tilde{\mathsf{C}}_{i,d}, \widehat{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{GSim}(1^{\mathsf{w}}, 1^h, \mathsf{Wrap}(\mathsf{sk}_i, (\mathsf{st}_{i,d}^{(2)}, \mathsf{msg}_{i,d}^{(2)})))$.

B. Set $\overline{\mathsf{lab}}_{i,d} := (\widehat{\mathsf{lab}}_{i,d}, \widehat{\mathsf{lab}}_{i,d})$.

C. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathcal{R}((i, p_q^*[d], p_q[d]), (t, b))$.

D. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

(c) For $1 < d \leq m$, and $i \in H$,

i. If $d < m$, then compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$, else, compute $(\tilde{\mathsf{C}}_{i,m}, \overline{\mathsf{lab}}_{i,m}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)))$.

ii. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell], b\in\{0,1\}}$, where each $K_{i,t,b} := \mathsf{PRF}(\mathsf{H}_{p_q[d]}(r_{i,d-1}), (t,b))$.

iii. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

iv. Compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$.

$\underline{\mathsf{Hyb}_7^{\mathcal{A}}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}_7^{\mathcal{A}}$, for all honest parties $i \in H$, circuit queries $q \in [Q]$ and depth $d = 1$, we replace the encryption of $\mathsf{est}_{i,1}^{(2)}$ with an encryption of zero, i.e $\mathsf{enc}(\mathsf{sk}_i, \vec{0})$. Indistinguishability follows from semantic security of enc. During the security reduction, there will be $|H|Q$ challenge ciphertexts. In step 10.(b).(ii), we do not decrypt $\mathsf{est}_{i,d}^{(2)}$, and instead use $\mathsf{st}_{i,d}^{(2)}$ from step 8.(b).(v).

7. Compute $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$ as follows.

(a) For each $i \in I, c \in \{0,1\}$,

i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,0})))$.

ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathsf{PRF}(\mathsf{H}_c(r_{i,0}), (t, z_{i,1}^c[t]))$.

iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.

(b) For each $i \in H, c \in \{0,1\}$,

i. Compute $z_{i,1}^c := (\mathsf{st}_{i,1,c}^{(1)}, \mathsf{msg}_{i,1,c}^{(1)}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathcal{U}(i, p_q^*[1], c)))$.

ii. For each $t \in [\ell]$, set $K_{i,t}^c := \mathcal{R}((i, p_q^*[1], c), (t, 0))$.

iii. Set $\widehat{K}_{i,1}^c := \{K_{i,t}^c\}_{t\in[\ell]}$.

(c) Provide $\begin{pmatrix} \widehat{K}_{1,1}^0 \dots \widehat{K}_{n,1}^0 \\ \widehat{K}_{1,1}^1 \dots \widehat{K}_{n,1}^1 \end{pmatrix}$ to $\mathsf{FMS.Sim}_0$, who responds with $\{\mathsf{msg}_{i,0}^{(2)}\}_{i\in H}$.

8. Compute $\left\{\{\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right\}_{i\in H}$ as follows.

(a) Skip.

(b) For $d = 1$,

i. For each $i \in H$, set $r_{i,d} := \mathcal{U}(i, p_q^*[d], p_q[d])$.

ii. Skip.

iii. Skip.

iv. Skip.

v. Set $(\mathsf{st}_{i,d}^{(2)}, \mathsf{msg}_{i,d}^{(2)}) \leftarrow \mathsf{MPC}_2(1^\lambda, \mathsf{CRS}, i, \mathsf{st}_{i,d,p_q[d]}^{(1)}, \{\mathsf{msg}_{j,d,p_q[d]}^{(1)}\}_{j\in[n]})$.

vi. Skip.

vii. For each $i \in H$,

A. Set $\mathsf{est}_{i,d}^{(2)} := \mathsf{enc}(\mathsf{sk}_i, \vec{0})$ and compute $(\tilde{\mathsf{C}}_{i,d}, \widehat{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{GSim}(1^\mathsf{w}, 1^h, (\mathsf{est}_{i,d}^{(2)}, \mathsf{msg}_{i,d}^{(2)}))$.

B. Set $\overline{\mathsf{lab}}_{i,d} := (\widehat{\mathsf{lab}}_{i,d}, \widehat{\mathsf{lab}}_{i,d})$.

C. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell], b\in\{0,1\}}$, where each $K_{i,t,b} := \mathcal{R}((i, p_q^*[d], p_q[d]), (t,b))$.

D. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

(c) For $1 < d \leq m$, and $i \in H$,

i. If $d < m$, then compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$,
else, compute $(\tilde{\mathsf{C}}_{i,m}, \overline{\mathsf{lab}}_{i,m}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)))$.

ii. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathsf{PRF}(\mathsf{H}_{p_q[d]}(r_{i,d-1}), (t,b))$.

iii. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

iv. Compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$.

10. Compute $\{y_i\}_{i\in H}$ as follows.

(a) Provide the $\{\mathsf{msg}^{(2)}_{i,0}\}_{i\in I}$ to $\mathsf{FMS.Sim}_0$. If $\mathsf{FMS.Sim}_0$ outputs $\mathsf{continue}$ then continue and otherwise set $\{y_i\}_{i\in H} := \perp$ and skip the next step.

(b) For $d = 1$, and $i \in H$,

i. For each $j \in [n]$,

A. Compute $\widehat{\mathsf{lab}}_{j,d} := \mathsf{LabDec}(\widehat{K}^{p_q[d]}_{j,d}, \overline{\mathsf{elab}}_{j,d})$.

B. Evaluate $(\mathsf{est}^{(2)}_{j,d}, \mathsf{msg}^{(2)}_{j,d}) := \mathsf{GEval}(\tilde{\mathsf{C}}_{j,d}, \widehat{\mathsf{lab}}_{j,d}))$.

ii. Skip.

iii. if $d < m$, compute $\begin{pmatrix} \widehat{K}^0_{1,d} \dots \widehat{K}^0_{n,d} \\ \widehat{K}^1_{1,d} \dots \widehat{K}^1_{n,d} \end{pmatrix} := \mathsf{MPC}_3(\mathsf{st}^{(2)}_{i,d}, \{\mathsf{msg}^{(2)}_{j,d}\}_{j\in[n]})$,
else, compute $y_i := \mathsf{MPC}_3(\mathsf{st}^{(2)}_{i,d}, \{\mathsf{msg}^{(2)}_{j,d}\}_{j\in[n]})$.

(c) For $1 < d \le m$, and $i \in H$,

i. For each $j \in [n]$,

A. Compute $\widehat{\mathsf{lab}}_{j,d} := \mathsf{LabDec}(\widehat{K}^{p_q[d]}_{j,d}, \overline{\mathsf{elab}}_{j,d})$.

B. Evaluate $(\mathsf{est}^{(2)}_{j,d}, \mathsf{msg}^{(2)}_{j,d}) := \mathsf{GEval}(\tilde{\mathsf{C}}_{j,d}, \widehat{\mathsf{lab}}_{j,d}))$.

ii. Decrypt $\mathsf{st}^{(2)}_{i,d} := \mathsf{dec}(\mathsf{sk}_i, \mathsf{est}^{(2)}_{i,d})$.

iii. if $d < m$, compute $\begin{pmatrix} \widehat{K}^0_{1,d} \dots \widehat{K}^0_{n,d} \\ \widehat{K}^1_{1,d} \dots \widehat{K}^1_{n,d} \end{pmatrix} := \mathsf{MPC}_3(\mathsf{st}^{(2)}_{i,d}, \{\mathsf{msg}^{(2)}_{j,d}\}_{j\in[n]})$,
else, compute $y_i := \mathsf{MPC}_3(\mathsf{st}^{(2)}_{i,d}, \{\mathsf{msg}^{(2)}_{j,d}\}_{j\in[n]})$.

$\underline{\mathsf{Hyb}^{\mathcal{A}}_8(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}^{\mathcal{A}}_8$ we simulate the MPC at depth $d = 1$ for all circuit queries $q \in [Q]$. In total, Q MPCs are simulated. Indistinguishability follows from the security of $\mathsf{FMS.MPC}$.

7. Compute $\{\mathsf{msg}^{(2)}_{i,0}\}_{i\in H}$ as follows.

(a) For each $i \in I, c \in \{0,1\}$,

i. Compute $z^c_{i,1} := (\mathsf{st}^{(1)}_{i,1,c}, \mathsf{msg}^{(1)}_{i,1,c}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,0})))$.

ii. For each $t \in [\ell]$, set $K^c_{i,t} := \mathsf{PRF}(\mathsf{H}_c(r_{i,0}), (t, z^c_{i,1}[t]))$.

iii. Set $\widehat{K}^c_{i,1} := \{K^c_{i,t}\}_{t\in[\ell]}$.

(b) For each $i \in H, c \in \{0,1\}$,

i. Skip.

ii. For each $t \in [\ell]$, set $K^c_{i,t} := \mathcal{R}((i, p^*_q[1], c), (t, 0))$.

iii. Set $\widehat{K}^c_{i,1} := \{K^c_{i,t}\}_{t\in[\ell]}$.

(c) Provide $\begin{pmatrix} \widehat{K}^0_{1,1} \dots \widehat{K}^0_{n,1} \\ \widehat{K}^1_{1,1} \dots \widehat{K}^1_{n,1} \end{pmatrix}$ to $\mathsf{FMS.Sim}_0$, who responds with $\{\mathsf{msg}^{(2)}_{i,0}\}_{i\in H}$.

8. Compute $\left\{\{\widetilde{\mathsf{C}}_{i,d}, \overline{\mathsf{elab}}_{i,d}\}_{d\in[m]}\right\}_{i\in H}$ as follows.

   (a) Skip.

   (b) For $d = 1$,

      i. For each $i \in I$, compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$, and for each $i \in H$, set $r_{i,d} := \mathcal{U}(i, p_q^*[d], p_q[d])$.

      ii. Initialize $\mathsf{FMS.Sim}$ with inputs $(1^\lambda, z, I, \{x_i, r_{i,d}\}_{i\in I})$. Denote this instance of the simulator by $\mathsf{FMS.Sim}_d$. Run $\mathsf{FMS.Sim}_d$ to obtain $\left(\mathsf{CRS}, \{\mathsf{msg}^{(1)}_{i,d}\}_{i\in H}\right)$.

      iii. Ignore $\mathsf{FMS.Sim}_d$'s output, and send back $\{\mathsf{msg}^{(1)}_{i,d,p_q[d]}\}_{i\in I}$.

      iv. $\mathsf{FMS.Sim}_d$ queries $\{\widetilde{x}_i, \widetilde{r}_{i,d}\}_{i\in I}$ to its ideal functionality, which is ignored.

      v. Compute as follows.

         A. For each $i \in [n], c \in \{0, 1\}$,

            - Compute $z^c_{i,d+1} := (\mathsf{st}^{(1)}_{i,d+1,c}, \mathsf{msg}^{(1)}_{i,d+1,c}) \leftarrow \mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, i, (x_i, \mathsf{G}_c(r_{i,d})))$.
            - For each $t \in [\ell]$, set $K^c_{i,t} := \mathsf{PRF}(\mathsf{H}_c(r_{i,d}), (t, z^c_{i,d+1}[t]))$.
            - Set $\widehat{K}^c_{i,d+1} := \{K^c_{i,t}\}_{t\in[\ell]}$.

         B. Provide $\begin{pmatrix} \widehat{K}^0_{1,d+1} \cdots \widehat{K}^0_{n,d+1} \\ \widehat{K}^1_{1,d+1} \cdots \widehat{K}^1_{n,d+1} \end{pmatrix}$ to $\mathsf{FMS.Sim}_d$, who responds with $\{\mathsf{msg}^{(2)}_{i,d}\}_{i\in H}$.

      vi. Skip.

      vii. For each $i \in H$,

         A. Set $\mathsf{est}^{(2)}_{i,d} := \mathsf{enc}(\mathsf{sk}_i, \vec{0})$ and compute $(\tilde{\mathsf{C}}_{i,d}, \widehat{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{GSim}(1^\mathsf{w}, 1^h, (\mathsf{est}^{(2)}_{i,d}, \mathsf{msg}^{(2)}_{i,d}))$.

         B. Set $\overline{\mathsf{lab}}_{i,d} := (\widehat{\mathsf{lab}}_{i,d}, \widehat{\mathsf{lab}}_{i,d})$.

         C. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathcal{R}((i, p_q^*[d], p_q[d]), (t, b))$.

         D. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

   (c) For $1 < d \le m$, and $i \in H$,

      i. If $d < m$, then compute $(\tilde{\mathsf{C}}_{i,d}, \overline{\mathsf{lab}}_{i,d}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{N}, \cdot, \cdot)))$,
      else, compute $(\tilde{\mathsf{C}}_{i,m}, \overline{\mathsf{lab}}_{i,m}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{Wrap}(\mathsf{sk}_i, \mathsf{MPC}_2(\mathsf{C}, \cdot, \cdot)))$.

      ii. Set $\overline{K}_{i,d} := \{K_{i,t,b}\}_{t\in[\ell],b\in\{0,1\}}$, where each $K_{i,t,b} := \mathsf{PRF}(\mathsf{H}_{p_q[d]}(r_{i,d-1}), (t, b))$.

      iii. Compute $\overline{\mathsf{elab}}_{i,d} \leftarrow \mathsf{LabEnc}(\overline{K}_{i,d}, \overline{\mathsf{lab}}_{i,d})$.

      iv. Compute $r_{i,d} := \mathsf{G}_{p_q[d]}(r_{i,d-1})$.

10. Compute $\{y_i\}_{i\in H}$ as follows.

   (a) Provide the $\{\mathsf{msg}^{(2)}_{i,0}\}_{i\in I}$ to $\mathsf{FMS.Sim}_0$. If $\mathsf{FMS.Sim}_0$ outputs $\mathsf{continue}$ then continue and otherwise set $\{y_i\}_{i\in H} := \perp$ and skip the next step.

   (b) For $d = 1$,

      i. For each $j \in [n]$,

         A. Compute $\widehat{\mathsf{lab}}_{j,d} := \mathsf{LabDec}(\widehat{K}^{p_q[d]}_{j,d}, \overline{\mathsf{elab}}_{j,d})$.

         B. Evaluate $(\mathsf{est}^{(2)}_{j,d}, \mathsf{msg}^{(2)}_{j,d}) := \mathsf{GEval}(\tilde{\mathsf{C}}_{j,d}, \widehat{\mathsf{lab}}_{j,d}))$.

      ii. Provide the $\{\mathsf{msg}^{(2)}_{i,d}\}_{i\in I}$ to $\mathsf{FMS.Sim}_d$. If $\mathsf{FMS.Sim}_d$ outputs $\mathsf{continue}$ then continue and otherwise set $\{y_i\}_{i\in H} := \perp$ and skip to step 11.

   (c) For $1 < d \le m$, and $i \in H$,

      i. For each $j \in [n]$,

         A. Compute $\widehat{\mathsf{lab}}_{j,d} := \mathsf{LabDec}(\widehat{K}^{p_q[d]}_{j,d}, \overline{\mathsf{elab}}_{j,d})$.

B. Evaluate $(\mathsf{est}^{(2)}_{j,d}, \mathsf{msg}^{(2)}_{j,d}) := \mathsf{GEval}(\tilde{\mathsf{C}}_{j,d}, \widehat{\mathsf{lab}}_{j,d}))$.

ii. Decrypt $\mathsf{st}^{(2)}_{i,d} := \mathsf{dec}(\mathsf{sk}_i, \mathsf{est}^{(2)}_{i,d})$.

iii. if $d < m$, compute $\begin{pmatrix} \widehat{K}^0_{1,d} \dots \widehat{K}^0_{n,d} \\ \widehat{K}^1_{1,d} \dots \widehat{K}^1_{n,d} \end{pmatrix} := \mathsf{MPC}_3(\mathsf{st}^{(2)}_{i,d}, \{\mathsf{msg}^{(2)}_{j,d}\}_{j\in[n]})$,
else, compute $y_i := \mathsf{MPC}_3(\mathsf{st}^{(2)}_{i,d}, \{\mathsf{msg}^{(2)}_{j,d}\}_{j\in[n]})$.

Observe that $\mathsf{Hybrid}^{\mathcal{A}}_8$ is exactly $\mathsf{r.Sim}^{\mathcal{A}}_1$ (assuming that $m > 1$, so that steps 8.(b).(vi) and 10.(b).(iv) are not invoked).

We iterate over following hybrids for $\xi \in [m]$ until we obtain $\mathsf{r.Sim}_m$.

$\underline{\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-7)}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-7)}$ we change the ξth level of randomness (of the PRG) for honest parties to uniform. This hybrid has a similar flavor as Hyb_2 and is therefore omitted.

$\underline{\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-6)}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-6)}$ we change the PRFs for honest parties at depth $(\xi+1)$ to uniform. This hybrid has a similar flavor as Hyb_3 and is therefore omitted.

$\underline{\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-5)}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-5)}$ we copy the labels of the honest parties at depth $(\xi+1)$. This hybrid has a similar flavor as Hyb_4 and is therefore omitted.

$\underline{\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-4)}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-4)}$ we make the keys $K^c_{i,t}$ of the honest parties at depth $(\xi+1)$ independent of $z^c_{i,\xi+1}[t]$ and hence of $(\mathsf{st}^{(1)}_{i,\xi+1,c}, \mathsf{msg}^{(1)}_{i,\xi+1,c})$ of MPC_1. This hybrid has a similar flavor as Hyb_5 and is therefore omitted.

$\underline{\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-3)}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-3)}$ we simulate the honest parties' garbled circuits at depth $(\xi+1)$. This hybrid has a similar flavor as Hyb_6 and is therefore omitted.

$\underline{\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-2)}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-2)}$ we replace for all $i \in H$, $\mathsf{est}^{(2)}_{i,\xi+1}$ with an encryption of zero, i.e $\mathsf{enc}(\mathsf{sk}_i, \vec{0})$. This hybrid has a similar flavor as Hyb_7 and is therefore omitted.

$\underline{\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-1)}(1^\lambda, z, I, \{x_i\}_{i\in[n]}):}$ In Hybrid $\mathsf{Hyb}^{\mathcal{A}}_{9+(7\xi-1)}$ we initialize and use $\mathsf{FMS.Sim}_{\xi+1}$ to generate for all $i \in H$, $\mathsf{msg}^{(2)}_{i,\xi+1}$. This hybrid has a similar flavor as Hyb_8 and is therefore omitted.

References

[ABJ+19] Prabhanjan Ananth, Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. From FE combiners to secure MPC and back. In *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part I*, pages 199–228, 2019.

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

[AMN+18] Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for NC^1 in traditional groups. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 543–574. Springer, Heidelberg, August 2018.

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2019, Part III*, LNCS, pages 489–518. Springer, Heidelberg, August 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 18*, pages 896–912. ACM Press, October 2018.

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

[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, April / May 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.

[BJOV18] Saikrishna Badrinarayanan, Abhishek Jain, Rafail Ostrovsky, and Ivan Visconti. Non-interactive secure computation from one-way functions. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 118–138. Springer, Heidelberg, December 2018.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Heidelberg, April / May 2018.

[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.

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

[CDI+19] Melissa Chase, Yevgeniy Dodis, Yuval Ishai, Daniel Kraschewski, Tianren Liu, Rafail Ostrovsky, and Vinod Vaikuntanathan. Reusable non-interactive secure computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2019, Part III*, LNCS, pages 462–488. Springer, Heidelberg, August 2019.

[DG17a] Nico Döttling and Sanjam Garg. From selective IBE to full IBE and selective HIBE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 372–408. Springer, Heidelberg, November 2017.

[DG17b] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 537–569. Springer, Heidelberg, August 2017.

[DGH+19] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. Two-round oblivious transfer from cdh or lpn. Cryptology ePrint Archive, Report 2019/414, 2019. `http://eprint.iacr.org/2019/414`.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Heidelberg, August 2016.

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

[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.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, October 1984.

[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.

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

[GIS18] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-round MPC: Information-theoretic and black-box. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 123–151. Springer, Heidelberg, November 2018.

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

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

[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.

[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.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[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. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499. Springer, Heidelberg, April / May 2018.

[IKO+11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 406–425. Springer, Heidelberg, May 2011.

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

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

[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.

[MR17] Payman Mohassel and Mike Rosulek. Non-interactive secure 2PC in the offline/online and batch settings. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 425–455. Springer, Heidelberg, April / May 2017.

[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.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997.

[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.

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