Reusable Two-Round MPC from LPN

James Bartusek[*,†] Sanjam Garg[‡,†] Akshayaram Srinivasan[§] Yinuo Zhang[¶]

March 10, 2021

Abstract

We present a new construction of maliciously-secure, two-round multiparty computation (MPC) in the CRS model, where the first message is reusable an unbounded number of times. The security of the protocol relies on the Learning Parity with Noise (LPN) assumption with inverse polynomial noise rate $1/n^{1-\epsilon}$ for small enough ϵ, where n is the LPN dimension. Prior works on reusable two-round MPC required assumptions such as DDH or LWE that imply some flavor of homomorphic computation. We obtain our result in two steps:

- In the first step, we construct a two-round MPC protocol in the *silent pre-processing model* (Boyle et al., Crypto 2019). Specifically, the parties engage in a computationally inexpensive setup procedure that generates some correlated random strings. Then, the parties commit to their inputs. Finally, each party sends a message depending on the function to be computed, and these messages can be decoded to obtain the output. Crucially, the complexity of the pre-processing phase and the input commitment phase do not grow with the size of the circuit to be computed. We call this *multiparty silent NISC*, generalizing the notion of two-party silent NISC of Boyle et al. (CCS 2019). We provide a construction of multiparty silent NISC from LPN in the random oracle model.
- In the second step, we give a transformation that removes the pre-processing phase and use of random oracle from the previous protocol. This transformation additionally adds (unbounded) reusability of the first round message, giving the first construction of reusable two-round MPC from the LPN assumption. This step makes novel use of randomized encoding of circuits (Applebaum et al., FOCS 2004) and a variant of the "tree of MPC messages" technique of Ananth et al. and Bartusek et al. (TCC 2020).

1 Introduction

Consider a scenario where a consortium of oncologists wants to compute several statistical tests on the confidential genomic data of their patients, while preserving the privacy of their patients. To accomplish this, each oncologist first publishes an encryption of their private database on their website. Next, given a proposed hypothesis F, the oncologists would like to figure out if this hypothesis is consistent with their joint databases. They would like to achieve this by sending a single message (that could grow with the size of the circuit computing F) to each other. Can they achieve this? What if they want to continue computing multiple hypotheses on the same data? Can they perform multiple tests at varying points in time while sending just one additional message for every new test? In other words, can they *reuse* the published encryptions of their data across multiple tests?

[*]University of California, Berkeley, `bartusek.james@gmail.com`

[†]Supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by the Sloan Foundation, Visa Inc., and Center for Long-Term Cybersecurity (CLTC, UC Berkeley). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

[‡]University of California, Berkeley and NTT Research, `sanjamg@berkeley.edu`.

[§]Tata Institute of Fundamental Research, `akshayaram.srinivasan@tifr.res.in`

[¶]University of California, Berkeley, `yinuo@berkeley.edu`

This scenario is a special case of the more general problem of constructing *reusable* two-round multiparty computation, whose feasibility was established in the work of Garg et al. [GGHR14] assuming the existence of indistinguishability obfuscation [BGI+01, GGH+13]. Starting with this work, an important line of research has been to weaken the computational assumptions required for constructing this primitive. The work of Mukherjee and Wichs [MW16] and a recent work of Ananth et al. [AJJM20] gave a construction from the Learning with Errors assumption [Reg05]. The work of Benhamouda and Lin [BL20] constructed such a protocol from standard assumptions on bilinear maps and the work of Bartusek et al. [BGMM20] provided a construction based on the DDH assumption.

Despite significant progress, our understanding of the assumptions necessary to realize two-round MPC protocols with reusability still lags behind the assumptions known to be sufficient for two-round MPC without reusability. In particular, while we know two-round MPC from any two-round OT [BL18, GS18a], known constructions of two-round MPC with reusability seem to require assumptions that support homomorphic computation — namely, LWE and DDH (which are known to imply various flavours of *homomorphic secret sharing* [BGI16]). In particular, these assumptions are known to imply some notion of *communication-efficient*[1] secure computation for a rich class of functions [MW16, BGI16, DHRW16]. In this work, we ask:

Can we realize reusable two-round MPC from assumptions that are not known to imply communication-efficient secure computation?

1.1 Our Results

We answer the above question in the affirmative by constructing a reusable two-round MPC protocol from the LPN assumption over binary fields with inverse polynomial noise rate $1/n^{1-\epsilon}$ for small enough constant ϵ, where n is the LPN dimension.

Our construction proceeds in two steps:

- **Multiparty Silent NISC:** We first consider the problem of constructing a two-round MPC protocol where the first round message is *succinct* (i.e., the complexity of computing the first round message does not grow with the circuit size) in the silent pre-processing model [BCG+19b]. To give more details, there is a pre-processing phase run by a dealer that generates correlated random strings for each party. In the first round, the parties send a commitment to their inputs using the correlated randomness. In the second round, the parties send a message that can be later decoded to obtain the output of the function. For efficiency, we require the complexity of the pre-processing phase and the input commitment phase to be independent of the circuit size and only the second round computation can depend on this parameter. We call this *multiparty Silent NISC*, and this naturally extends a similar notion defined by Boyle et al. [BCG+19a] for the two-party case. We give a construction of a multiparty silent NISC protocol with silent pre-processing in the random oracle model based on the LPN assumption.
- **Reusable Two-Round MPC:** In the second step, we transform the above protocol to a protocol in the CRS model that achieves unbounded reusability without increasing the number of rounds or requiring stronger assumptions. As a corollary, we obtain the first construction of reusable two-round MPC in the CRS model from the LPN assumption.

2 Technical Overview

In this section, we first discuss the notion of multiparty non-interactive secure computation with silent preprocessing, which is a natural extension of the silent NISC primitive of [BCG+19a] to the multiparty setting. We then give an overview of our construction of multiparty silent NISC from the LPN assumption

[1]By communication efficiency, we mean that the communication cost of the protocols do not grow with the circuit size of the functionality to be computed.

in the random oracle model. This result mostly follows from a combination of ideas from [GIS18] and [BCG$^+$19b], with a few necessary tweaks. Finally, we give an overview of the transformation from multiparty NISC to reusable two-round MPC. This transformation forms the heart of our technical contribution.

2.1 Multi-party Silent NISC

In a silent NISC protocol [BCG$^+$19a], two parties begin by interacting in a preprocessing phase that results in some shared correlated randomness. In addition, they send to each other encodings of their inputs x and y. So far, all computation and communication is "small", i.e. it does not grow with the size of the circuit C they will eventually want to compute on their inputs. At this point, one party may publish a *single* (large) message to the other party, allowing the latter to learn the value $C(x, y)$. Since all communication before this point was small, the parties will be required to "silently" expand their correlated randomness into useful correlations needed for the final non-interactive computation phase.

We naturally extend this interaction pattern to the multi-party setting. We outline a three-phase approach for computing a m-party functionality.

- *Preprocessing phase:* A trusted dealer computes correlated secrets $\{s_i\}_{i\in[m]}$ and sends s_i to party i.
- *Input commitment phase:* Party i, using secret s_i, computes and broadcasts a commitment c_i to its input x_i.
- *Compute phase:* Once a circuit C is known to all parties, they each compute and broadcast a single message m_i.
- *Recovery:* The protocol is publicly decodable. That is, the messages $\{m_i\}_{i\in[m]}$ can be combined by any party (inside or outside the system) to recover the output $Y \leftarrow C(x_1, \ldots, x_n)$.

Crucially, we require the computation and communication during the pre-processing and input commitment phases to only grow as a fixed polynomial in the input size and the security parameter, and not with the size of C (although an upper bound on the size of supported circuits may be known during these phases).

2.1.1 Starting point: PCG

Based on prior works, we can construct a multiparty NISC protocol using either a multi-key fully-homomorphic encryption [MW16, DHRW16], or homomorphic secret sharing [BGMM20], or using a specialized type of witness encryption [BL20, GS17]. However, each of these approaches make use of assumptions that can support some (limited) form of homomorphic computation on encrypted data. Further, these protocols have a fairly inefficient compute phase. For example, the approach of [BGMM20] requires the parties to compute a PRF homomorphically under a HSS scheme - this non-black-box use of cryptography will be prohibitively inefficient in practice.

On the other hand, the works of [BCG$^+$19b] and [BCG$^+$19a] study methods for distributing short seeds to two parties which can then be silently and efficiently expanded into useful two-party correlations under the LPN assumption. For example, they show how to generate many random oblivious transfer (OT) correlations efficiently via short seeds, and they call the primitive that accomplishes this a *pseudorandom correlation generator* (PCG) for OT correlations.

Now, given pairwise random OT correlations between each pair of parties, [GIS18] shows how to implement the two-round MPC protocol of [GS18b] (which we refer to as GS18) in a black-box manner. Their approach would fit the template of multi-party silent NISC, except that their input commitment phase would also grow with the size of the circuit, and thus the resulting protocol would not be "succinct". In this work, we show how to use the PCG techniques of [BCG$^+$19b, BCG$^+$19a] in order to generate more sophisticated correlations that suffice to instantiate GS18 while keeping the input commitment phase independent of the circuit to be computed.

2.1.2 A PCG for GS18 Correlations

In [GIS18], the random OT correlations and first-round messages (which also function as input commitments) are essentially used to set up certain *structured* OT correlations that enable the parties to compute a circuit over their joint inputs with only one additional message. At a high level, these structured correlations allow parties to each output sequences of garbled circuits that communicate with each other in order to implement an MPC protocol among themselves, though the details of this will not be important for this discussion. Here, we directly describe the correlation which consists of pairwise correlations set up between each pair of parties, one acting as a sender and one as a receiver. The sender gets random OT messages $\{(m_{t,0}, m_{t,1})\}_{t\in[T]}$ and the receiver gets a random string v along with messages $\{m_{t,z_t}\}_{t\in[T]}$. Each z_t is *not* a uniformly random and independent bit, rather, each is computed as $z_t = \mathsf{NAND}\left(\mathsf{v}[f] \oplus \alpha, \mathsf{v}[g] \oplus \beta\right) \oplus \mathsf{v}[h]$ for some indices (f, g, h) and constants (α, β).

As we will see below, one can write what is described so far as a two-party bilinear correlation. This is good news, since the work of [BCG+19b] constructed a PCG for two-party bilinear correlations. However, we do not generically make use of their PCG, for two reasons. First, we will actually require a *multi-party* correlation, since each party's random string v must be shared among all of the two-party correlations it sets up with each other party. Next, we have a more stringent requirement on the complexity of expansion. In particular, parties must use some of their expanded randomness in the input commitment phase, which must be efficient. We set up the PCG so that parties can obtain some part of the expanded randomness without expanding the entire set of correlations, which is computation that would grow with the size of the circuit. Thus, we describe how to set up the multi-party correlations necessary for GS18 from basic building blocks. Although our construction and proof follow those of [BCG+19b] very closely, we give a full description of the scheme in the body for the sake of completeness.

Now we briefly review the PCG of [BCG+19b, BCG+19a] that produces a large number of (unstructured) random OT correlations. Fix parameters $n' > n$. The dealer first samples a sparse binary error vector $y \in \mathbb{F}_2^{n'}$ (with a compact description denoted by $\widetilde{y}$) and a random offset (shift) $\delta \in \mathbb{F}_{2^\lambda}$. Then, $y \cdot \delta$ is secret shared into shares k_0, k_1, which are vectors in $\mathbb{F}_{2^\lambda}^{n'}$ and also have compact descriptions $\widetilde{k}_0, \widetilde{k}_1$ (this step requires the use of Distributed Point Functions [GI14]). Finally, $(\widetilde{k}_0, \widetilde{y})$ is given to the receiver, $(\widetilde{k}_1, \delta)$ is given to the sender, and a n'-by-n random binary matrix H is made public. In order to expand these short seeds into n random OT correlations, the receiver first expands its compact descriptions into (k_0, y) and then computes $t_0 := k_0 \cdot H \in \mathbb{F}_{2^\lambda}^n$ and $z := y \cdot H \in \mathbb{F}_2^n$, and the sender expands its compact description into k_1 and computes $t_1 := k_1 \cdot H \in \mathbb{F}_{2^\lambda}^n$. It is easy to check that $t_0 = t_1 + z \cdot \delta$ and thus for each $i \in [n]$, $(z[i], t_0[i])$, $(t_1[i], t_1[i] + \delta)$ is a correlated random OT instance. The choice bits z are random due to the LPN assumption. In order to remove the correlated offset δ, the parties can use a correlation robust hash function [IKNP03] or a random oracle, to hash each OT string.

Recall that in our setting, we actually require some structure on the string z of choice bits. To implement this, we first write each expression $z_t = \mathsf{NAND}(\mathsf{v}[f] \oplus \alpha, \mathsf{v}[g] \oplus \beta) \oplus \mathsf{v}[h]$ as a degree-two equation over $\mathbb{F}_2$ whose variables are entries of v. That is, $z_t = \mathsf{v}[f]\mathsf{v}[g] + \alpha\mathsf{v}[g] + \beta\mathsf{v}[f] + \mathsf{v}[h] + \alpha\beta + 1$. In order to obtain these degree-two correlated OT, we follow the construction of PCGs for constant-degree relations from [BCG+19b]. In particular, we define the error vector to be $y' := (1, y) \otimes (1, y) \in \mathbb{F}_2^{n' \cdot n'}$. Same as before, y' is secret shared into $k_0, k_1 \in \mathbb{F}_{2^\lambda}^{n' \cdot n'}$. Now, the receiver can compute $\mathsf{v} := y \cdot H$ and set $z := (1, \mathsf{v}) \otimes (1, \mathsf{v}) \in \mathbb{F}_2^{n \cdot n}$, and likewise the receiver and sender can compute vectors $t_0 := k_0 \cdot (H' \otimes H')$ and $t_1 := k_1 \cdot (H' \otimes H')$ respectively, where H' is $\begin{pmatrix} 1 & \\ & H \end{pmatrix}$. Both are vectors in $\mathbb{F}_{2^\lambda}^{n \cdot n}$, such that for any $f, g \in [n]$ and any degree-one or degree-two monomial $\mathsf{v}[f]\mathsf{v}[g]$ over the entries of v, there exists an index i such that $\left(z[i] := \mathsf{v}[f]\mathsf{v}[g], t_0[i]\right), \left(t_1[i], t_1[i] + \delta\right)$ is a valid correlated OT. One can then obtain any degree-two correlated OT by taking appropriate linear combinations. Correctness of this step crucially relies on the fact that all the "base" correlated OTs have the same shift δ. After taking the linear combinations, the parties can still apply a correlation robust hash function to get structured OT correlations with random sender strings.

In the body, we show that even in the setting where there is one receiver with a fixed error vector y, but multiple senders with different random offsets δ_i, one can still show security via reverse sampleability. In particular, for any one of n parties, their output correlation can be reverse sampled, given the output correlations of all other parties.

2.1.3 The Final Protocol

Given ideas from the previous section, we can complete our description of multiparty silent NISC from LPN in the random oracle model.

In the preprocessing phase, a trusted dealer sets up pairwise structured OT correlations between each pair of parties as described above. We include a random oracle in the CRS, which is used to generate the (large) matrix H and also used as a correlation robust hash function. In the input commitment phase, we have parties partially expand their correlated seeds into randomness that may be used to mask their inputs. Crucially, this step does not require fully expanding their seeds into the entire set of structured correlations that will be used in the compute phase, so we maintain the "silent" notion. To implement this, we actually sample two different H_1, H_2 matrices and two different error vectors y_1, y_2 of different sizes, and set $v = (y_1, y_2)\begin{pmatrix} H_1 & \\ & H_2 \end{pmatrix}$. As long as each y_b has sufficient error positions, we can still rely on LPN with inverse polynomial error rate. Finally, in the compute phase, the parties publish GS18 second round messages computed with respect to their expanded correlations, thus completing the protocol. Since the GS18 protocol is publicly decodable, so is our protocol.

As a final note, we can remove the random oracle at the cost of having a large CRS. In particular, given a bound on the size of the circuit to be computed, we can instantiate the protocol with a CRS that contains the H matrix (note that the size of this matrix must grow with the number of OT correlations generated and thus, the size of the circuit to be computed). Although this CRS is large, it can be *reused* across any number of input commitment and compute phases - a property that we take advantage of in the next section, which focuses on a construction of reusable two-round MPC from LPN. We will also have to replace the use of the random oracle as a correlation-robust hash function. As already observed in [BCG+19b], the role of correlation robust hash function can be replaced by an encryption scheme which is semantically secure against related-key attack for the class of linear functions. It is known that such an encryption scheme can be based on the LPN assumption [AHI11].

2.2 Reusable Two-round MPC from LPN

We now turn to our main result - a reusable two-round MPC protocol from the LPN assumption. Our approach takes the multiparty silent NISC protocol from last section as a starting point and constructs from it a *first message succinct* two-round MPC (FMS-MPC). An FMS-MPC protocol satisfies the property that the size of computation and communication necessary in the first round only grows with the input size and security parameter, and not with the size of the circuit to be computed in the second round. The work of [BGMM20] shows that FMS-MPC implies reusable two-round MPC, so we appeal to their theorem to finish our construction. Our construction of FMS-MPC proceeds in two steps.

2.2.1 Step 1: Bounded FMS-MPC.

In order to convert a multiparty silent NISC protocol into a two-round MPC, we need to remove the preprocessing phase, instantiating the dealer's computation in a distributed manner. A natural approach is to use a two-round MPC (e.g. GS18) to compute the preprocessing and input commitment phases, and after this is completed, have the parties compute and send their compute phases messages. However, this results in a three-round MPC protocol.

To collapse this protocol into two rounds, we use an idea from [BGMM20] - the two-round MPC which implements the dealer will compute *garbled labels* corresponding to the outputs of the preprocessing and input commitment phases, and in the second round, parties will also release *garbled circuits* that output their compute phase messages. Anyone can then combine the garbled inputs and garbled circuits to learn the entire set of compute phase messages, which will then allow one to recover the output of the circuit.

Since the computation necessary for computing the preprocessing and input commitment phases is small, the first round of the resulting protocol is succinct.

However, recall that the multiparty silent NISC constructed in last section requires a *large* CRS if instantiated without the use of a random oracle. In an FMS-MPC, the size of CRS should only depend on the security parameter, not the circuit size. Thus, we do not quite obtain an FMS-MPC following the above approach. Rather, we obtain what we call a *bounded* FMS-MPC, which has a large (but reusable) CRS whose size grows with the size of the circuit to be computed. Meanwhile, this MPC protocol is bounded since the size of the CRS determines the bound on the circuit size that can be supported.

2.2.2 Step 2: From Bounded FMS-MPC to FMS-MPC.

Thus, our task is to reduce the size of the CRS as well as to enable computation of unbounded polynomial-size circuits in the second round. This forms the main technical contribution of the second step.

To support unbounded circuit size, our idea is to use a *randomized encoding* in order to break down the computation of one large circuit into the computation of many small circuits. In particular, using results from [AIK05] for example, one can compute any a priori unbounded polynomial-size circuit with a number of "small" circuits, where this number depends on the original circuit size. Here "small" means that the size of each individual circuit is some fixed polynomial in the security parameter. Thus, the size of the CRS required to compute each of these small circuits only grows with the security parameter. Moreover, the CRS in our bounded FMS-MPC protocol is reusable, so the same small CRS can be used to compute each small circuit of randomized encodings.

However, computing each of the small circuits in parallel does not result in an FMS-MPC. Indeed, to maintain security this would require a different first round message for computing each randomized encoding circuit, and thus the total size of first round messages will still grow with the original circuit size. To remedy this, we use a variant of the tree-based approach from [AJJM20, BGMM20]. We construct a polynomial-size tree of bounded FMS-MPC instances, where each internal node computes two sets of fresh first round messages which are to be used to compute its two child nodes. Each leaf node corresponds to one of the small randomized encoding circuits. The first round message in our final FMS-MPC protocol will only consist of the first round messages for computing the *root* of this tree. In the second round, parties release garbled circuits that compute the second round message for each node in this tree. As before, to assist evaluation of these garbled circuits, each node will instead output *garbled labels* corresponding to the second round messages. This allows anyone to evaluate the entire tree, eventually learning the outputs of each leaf MPC, thus learning the randomized encoding of the original circuit that was computed.

Crucially, the small CRS can be reused to compute each node of this tree, so that each internal node does not need to generate a fresh CRS for its children. This allows the tree to grow to some unbounded polynomial size without each node computation becoming prohibitively large - each node just computes two sets of first round messages of the bounded FMS-MPC, which in total has some fixed polynomial size. Additional details of this construction can be found in Section 5.2.

2.2.3 On the LPN Assumption.

In both the multiparty NISC and the reusable MPC results, we rely on LPN with inverse polynomial error rate. In both cases, the reason is that we need the computation in the first phase (pre-processing / first-round message) to be (polynomially) smaller than the size of the circuits supported in the second phase.[2] In the first phase, parties perform computation that sets up the error vector, so the number of error positions in the vector must be inverse polynomially related to the number of correlations eventually used by the parties, which roughly corresponds to the number of LPN samples. Thus, we can rely on LPN with noise rate $1/n^{1-\epsilon}$, see Section 3.5 below for more details.

[2]Note that, as discussed above, in the reusable MPC case, we only need to support circuits of some fixed polynomial size in order to allow parties to compute circuits of *unbounded* polynomial size.

3 Preliminaries

Let λ denote the security parameter. A function $\mu(\cdot)\colon \mathbb{N} \to \mathbb{R}$ is said to be negligible if for any polynomial poly $(\cdot)$ there exists $\lambda_0 \in \mathbb{N}$ such that for all $\lambda > \lambda_0$ we have $\mu(\lambda) < \frac{1}{\mathrm{poly}(\lambda)}$. We will use $\mathrm{negl}(\lambda)$ to denote an unspecified negligible function and $\mathrm{poly}(\lambda)$ to denote an unspecified polynomial function. We denote $[k]$ to be the set $\{1,\ldots,k\}$. For any n-dimensional vector y, we use $y[i]$ to denote the i^{th} entry of y, and use $y[:k]$ $(k \le n)$ to denote its truncation from $y[1]$ to $y[k]$. 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 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 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. It additionally ensures that the decryption either outputs the original message corresponding to some ciphertext c, or it outputs nothing. This is formalized by the following robustness property. For any PPT adversary $\mathcal{A}$ and message m,

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

A robust private-key encryption can be constructed from any PRF [BGMM20].

3.2 Garbled Circuits

Below we recall the definition of garbling scheme for circuits [Yao86]. 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:

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

A garbling scheme satisfies the following properties:

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

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

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

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

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

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

Remark 3.1. *In this paper we sometimes uses $\overline{\boldsymbol{lab}}$ to denote $\{\boldsymbol{lab}_{i,b}\}_{i\in[n],b\in\{0,1\}}$ (all input labels), and we use $\widehat{\boldsymbol{lab}}$ to denote $\{\boldsymbol{lab}_{i,x_i}\}_{i\in[n]}$ (a garbled input).*

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

$$\begin{aligned}
\left(\widetilde{C}, \{\mathbf{lab}_{i,x_i}\}_{i\in[n]}\right) &\leftarrow \mathsf{Sim}\left(1^{|C|}, 1^{|x|}, C(x)\right), \\
\{\mathbf{lab}'_{i,x_i}\}_{i\in[n]} &\leftarrow \mathcal{A}\left(\widetilde{C}, \{\mathbf{lab}_{i,x_i}\}_{i\in[n]}\right), \\
Y &\leftarrow \mathsf{GEval}\left(\widetilde{C}, \{\mathbf{lab}'_{i,x_i}\}_{i\in[n]}\right), \\
\left(\{\mathbf{lab}'_{i,x_i}\}_{i\in[n]} \neq \{\mathbf{lab}_{i,x_i}\}_{i\in[n]}\right) &\wedge (Y \neq \perp).
\end{aligned}$$

Remark 3.2. *One can add authenticity of input labels property generically to any garbled circuit construction by digitally signing every input label and including the verification key as part of the garbled circuit $\widetilde{C}$.*

Label encryption. We design a specific encryption scheme for encrypting a grid of $2 \times n$ input labels corresponding to any garbled circuits. This encryption scheme uses a grid of $2 \times n$ keys (denoted by $\overline{K}$). 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 3.1. It then randomly permutes each pair (column) of ciphertexts, and outputs the resulting $2 \times n$ grid. On the other hand, decryption only takes as input a set of n keys (denoted by $\widehat{K}$), 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 encryption scheme 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.

- $\overline{\mathbf{elab}} \leftarrow \mathsf{LabEnc}\left(\overline{K}, \overline{\mathbf{lab}}\right)$: On input a grid of $2 \times n$ keys $\overline{K} := \{K_{i,b}\}_{i\in[n],b\in\{0,1\}}$, and $\overline{\mathbf{lab}} := \{\mathbf{lab}_{i,b}\}_{i\in[n],b\in\{0,1\}}$, LabEnc draws n random bits $b'_i \leftarrow \{0,1\}$ and outputs $\overline{\mathbf{elab}} := \{\mathbf{elab}_{i,b}\}_{i\in[n],b\in\{0,1\}}$, where $\mathbf{elab}_{i,b} = \mathsf{rob.Enc}\left(K_{i,b\oplus b'_i}, \mathbf{lab}_{i,b\oplus b'_i}\right)$.
- $\widehat{\mathbf{lab}} \leftarrow \mathsf{LabDec}\left(\widehat{K}, \overline{\mathbf{elab}}\right)$: On input a set of n keys $\widehat{K} := \{K_i\}_{i\in[n]}$, a grid of $2 \times n$ encrypted labels $\overline{\mathbf{elab}} := \{\mathbf{elab}_{i,b}\}_{i\in[n],b\in\{0,1\}}$, it outputs $\widehat{\mathbf{lab}} := \{\mathbf{lab}_i\}_{i\in[n]}$, where for each $i \in [n]$, $\mathbf{lab}_i = \mathsf{rob.Dec}(K_i, \mathbf{elab}_{i,0})$ if it is not $\perp$ and $\mathsf{rob.Dec}(K_i, \mathbf{elab}_{i,1})$ otherwise.

3.3 Two-Round MPC

We consider the following syntax of two round MPC protocol.

Definition 3.3 (Two-Round MPC Protocol)**.** *An m-party two-round MPC protocol is described by a triplet of PPT algorithms $\pi := (\mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$ with the following syntax:*

- $\left(\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)}\right) \leftarrow \mathsf{MPC}_1\left(1^\lambda, \mathsf{CRS}, C, i, x_i, r_i\right)$*: Takes as input 1^λ, a common reference string* CRS *(whose size should only depend on λ; we drop mentioning it explicitly if not needed), (the description of) a circuit C to be computed, identity of a party $i \in [m]$, 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* $\mathsf{st}_i^{(1)}$*.*
- $\left(\mathsf{st}_i^{(2)}, \mathsf{msg}_i^{(2)}\right) \leftarrow \mathsf{MPC}_2\left(C,\ \mathsf{st}_i^{(1)}, \left\{\mathsf{msg}_j^{(1)}\right\}_{j\in[m]}\right)$*: Takes as input (the description of) a circuit C to be computed, the state of a party* $\mathsf{st}_i^{(1)}$*, and the first round messages of all the parties* $\left\{\mathsf{msg}_j^{(1)}\right\}_{j\in[m]}$*. It outputs party i's second round message* $\mathsf{msg}_i^{(2)}$ *and its private state* $\mathsf{st}_i^{(2)}$*.*

- $Y \leftarrow \mathsf{MPC}_3\left(\mathsf{st}_i^{(2)}, \left\{\mathsf{msg}_j^{(2)}\right\}_{j\in[m]}\right)$: *Takes as input the state of a party* $\mathsf{st}_i^{(2)}$*, and the second round messages of all the parties* $\left\{\mathsf{msg}_j^{(2)}\right\}_{j\in[m]}$*. It outputs the the output* $Y := C\left(x_1, \ldots, x_m\right)$*.*

Remark 3.4. *In order to distinguish this notion of two-round MPC from the other enhanced notions, we will sometimes refer to this notion of two round MPC as the vanilla notion of two round MPC.*

Remark 3.5. *In this work we often consider the vanilla two-round MPC to be publicly decodable. That is, anyone can recover the output* y *based on just* $\left\{\mathsf{msg}_j^{(2)}\right\}_{j\in[m]}$*. We drop mentioning* $\mathsf{st}_i^{(2)}$ *in the publicly decodable setting.*

3.3.1 Security

We follow the standard real/ideal world paradigm [Gol04]. Consider m parties $(P_1, \ldots, P_m)$ with inputs $(x_1, \ldots, x_m)$ 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 the circuit C) is defined by comparing the real-world execution of the protocol with an ideal-world evaluation of C by a trusted party (ideal functionality). Informally, it is required that for every adversary $\mathcal{A}$ that corrupts some subset of the parties $I \subset [m]$ and participates in the real execution of the protocol, there exist an adversary Sim, also referred to as the simulator, which can achieve the same effect in the ideal-world. Denote $\vec{x} := (x_1, \ldots, x_m)$. We now formally describe the security definition.

3.3.2 Real Execution

In the real execution, the protocol π for computing C is executed in the presence of an adversary $\mathcal{A}$. The adversary takes as input the security parameter λ, the CRS 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 following any arbitrary polynomial-time strategy. The interaction of $\mathcal{A}$ in the protocol defines a random variable $\mathsf{Real}_{\mathcal{A}}(\lambda, \vec{x}, z, I)$ whose value is determined by the coin tosses of the adversary and the honest parties. This random variable contains the output of the adversary as well as the outputs of the honest parties.

3.3.3 Ideal Execution

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

- **Honest parties send inputs to the trusted party**: Each honest party sends its input to the trusted party. Each corrupted party P_i 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 $Y \leftarrow C\left({x_1}', \ldots, {x_m}'\right)$ and sends Y to Sim.

- **Adversary decides whether to continue or abort**: After seeing the values y, the adversary sends either an abort or continue message to the honest parties. (A semi-honest adversary never aborts.)

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

Definition 3.6. *Let* λ *be the security parameter and let* C *be any* m *party functionality. We say that a two round MPC protocol* π *securely computes* C *in the presence of malicious/semi-honest adversaries if for every*

PPT real world malicious/semi-honest adversary $\mathcal{A}$, there exists a PPT ideal world adversary Sim*, such that for any $\vec{x} \in (\{0,1\}^*)^m$, $z \in \{0,1\}^*$, and $I \subset [m]$, and for any PPT distinguisher D, we have that:*

$$\Pr\left[\ D\left(\mathsf{Real}_{\mathcal{A}}\left(\lambda, \vec{x}, z, I\right)\right) = 1\ \right] - \Pr\left[\ D\left(\mathsf{Ideal}_{\mathsf{Sim}}\left(\lambda, \vec{x}, z, I\right)\right) = 1\ \right] = \mathrm{negl}(\lambda).$$

In the same setting, we also consider the notion of 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 3.7 (FMS-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 $\left(1^\lambda, \mathsf{CRS}, i, x_i\right)$.*

A first message succinct two-round MPC also satisfies the same correctness and security properties as the (vanilla) two-round MPC protocol.

3.4 Computational Randomized Encoding

We recall the following definition of computational randomized encoding in [AIK05] [BL20].

Definition 3.8 (Computational randomized encoding)**.**
Let $f = \left\{f_n : \{0,1\}^n \to \{0,1\}^{l(n)}\right\}_{n \in q(\lambda)}$ be a polynomial-time computable function, computed by the uniform circuit family $\{C_n\}_{n \in q(\lambda)}$. A computational randomized encoding scheme for f is a tuple of three polynomial-time algorithms $(\mathsf{CRE.Enc}, \mathsf{CRE.Dec}, \mathsf{CRE.Sim})$ *with the following syntax and properties:*

- $\{\widehat{f}_n\}_{n \in q(\lambda)} \leftarrow \mathsf{CRE.Enc}\left(1^\lambda, f\right)$*: on input the security parameter, a function f, it outputs computational randomized encoding of f, where each $\widehat{f}_n : \{0,1\}^n \times \{0,1\}^\lambda \to \{0,1\}^s$ will be an* NC^0 *circuit.* $\mathsf{CRE.Enc}$ *is deterministic.*
- $y \leftarrow \mathsf{CRE.Dec}\left(1^\lambda, f, \{\widehat{y}_n\}_{n \in q(\lambda)}\right)$*: on input the security parameter, a function f, and the output $\widehat{y}_n$ of each randomized encoding $\widehat{f}_n$, it outputs y.* $\mathsf{CRE.Dec}$ *is deterministic.*
- $\{\widetilde{y}_n\}_{n \in q(\lambda)} \leftarrow \mathsf{CRE.Sim}\left(1^\lambda, f, y\right)$*: on input the security parameter, a function f and an output $y \in \{0,1\}^{l(n)}$, it outputs simulated computational randomized encoding $\{\widetilde{y}_n\}_{n \in q(\lambda)}$.*

Correctness *: For every function $f = \left\{f_n : \{0,1\}^n \to \{0,1\}^{l(n)}\right\}_{n \in q(\lambda)}$, for every input $x \in \{0,1\}^n$ and for every bit string $r \in \{0,1\}^\lambda$,*

$$\Pr\left[\mathsf{CRE.Dec}\left(1^\lambda, f, \left\{\widehat{f}_n(x,r)\right\}_{n \in q(\lambda)}\right) = f(x) \mid \left\{\widehat{f}_n\right\}_{n \in q(\lambda)} = \mathsf{CRE.Enc}\left(1^\lambda, f\right)\right] = 1$$

Security *: For every function $f = \left\{f_n : \{0,1\}^n \to \{0,1\}^{l(n)}\right\}_{n \in q(\lambda)}$ and for every input $x \in \{0,1\}^n$, the following two distributions are computationally indistinguishable:*

$$\left\{\left\{\widehat{f}_n\right\}_{n \in q(\lambda)} = \mathsf{CRE.Enc}\left(1^\lambda, f\right), r \leftarrow \{0,1\}^\lambda : \left\{\widehat{f}_n(x,r)\right\}_{n \in q(\lambda)}\right\}$$
$$\overset{c}{\approx} \left\{\{\widetilde{y}_n\}_{n \in q(\lambda)} = \mathsf{CRE.Sim}\left(1^\lambda, f, f(x)\right)\right\}$$

Above definition of randomized encoding can be constructed from a variant of Yao's garbling scheme [Yao86]. We assume without loss of generality that each $\widehat{f}_n$ is a circuit of size $p(\lambda)$ where $p(\cdot)$ is a fixed polynomial in λ.

3.5 Learning Parity with Noise

We recall the decisional exact Learning Parity with Noise (LPN) assumption over binary fields. The word "exact" modifies the standard decisional Learning Parity with Noise problem by changing the sampling procedure for the error vector. Instead of setting each component of $e \in \mathbb{F}_2^n$ to be 1 with independent probability, we sample e uniformly from the set of error vectors with exactly t entries set to 1. We let $\mathcal{HW}_{n,t}$ denote the uniform distribution over binary strings of length n with Hamming weight t. The exact LPN problem is polynomially equivalent to the standard version following the search to decision reduction given in [AIK09], as noted in [JKPT12]. We give the precise definition in its dual formulation.

Definition 3.9 (Exact Learning Parity with Noise)**.** *Let λ be the security parameter and let $n(\cdot), n'(\cdot), t(\cdot)$ be some polynomials. The (dual) Decisional Exact Learning Parity with Noise problem with parameters $(n(\cdot), n'(\cdot), t(\cdot))$ is hard if, for every probabilistic polynomial-time algorithm $\mathcal{A}$, there exists a negligible function μ such that*

$$\left| \Pr_{B,e}[\mathcal{A}(B, e \cdot B) = 1] - \Pr_{B,u}[\mathcal{A}(B, u) = 1] \right| \leq \mu(n)$$

where $B \leftarrow \mathbb{F}_2^{n'(\lambda) \times n(\lambda)}, e \leftarrow \mathcal{HW}_{n'(\lambda),t(\lambda)}$, and $u \leftarrow \mathbb{F}_2^{n(\lambda)}$.

Throughout this work, we will use the following flavor of LPN assumption. For a given security parameter λ and polynomial $p(\lambda)$, we will need to assume that LPN is hard when e has Hamming weight λ and $e \cdot B$ is a vector of length $p(\lambda)$. Thus, we can set $n = p(\lambda)$ and $n' = 2n$, which corresponds to a (primal) LPN assumption of dimension n and error rate $\lambda/2n = 1/n^{1-\epsilon}$ for some constant ϵ. This is referred to as "LPN with inverse polynomial error rate".

3.6 PCG

We recall the following definition of PCG from [BCG+19b]:

Definition 3.10 (Reverse-sampleable Correlation Generator)**.** *Let C be a correlation generator, that is, $C(1^\lambda)$ outputs two random strings (R_0, R_1) according to some joint distribution. We say C is reverse sampleable if there exists a PPT algorithm* Rsample *such that for $b \in \{0,1\}$ the correlation obtained via:*

$$\{(R_0', R_1') \,|\, (R_0, R_1) \leftarrow C\left(1^\lambda\right), R_b' := R_b, R_{1-b}' \leftarrow \mathsf{Rsample}\left(b, R_b\right)\}$$

is indistinguishable from $\{(R_0, R_1) \leftarrow C\left(1^\lambda\right)\}$.

In this work, we primarily consider the following correlation generators:

- *Correlated OT:* $\left\{(R_0 := (\sigma, m_\sigma), R_1 := (m_0, m_1 := m_0 + \delta)) \leftarrow C(1^\lambda)\right\}$. Where δ is a random element in some field, each $\sigma \in \{0,1\}$ and each m_0 are uniformly sampled. This correlation generator is clearly reverse-sampleable. In this work we sometimes refer to R_0 as the receiver strings and R_1 as the sender strings.
- *Subfield-VOLE:* $\left\{(R_0 := (\vec{u}, \vec{v}), R_1 := (\delta, \vec{w})) \leftarrow C(1^\lambda)\right\}$. Where $(\vec{u}, \vec{v}) \in \mathbb{F}_2^n \times \mathbb{F}_{2^\lambda}^n$, $(\delta, \vec{w}) \in \mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^\lambda}^n$, and where $\vec{u}, \vec{v}$, and δ are uniformly random, and $\vec{v} = \vec{u}\delta + \vec{w}$. This correlation generator is also reverse-sampleable.

Definition 3.11 (Pseudorandom Correlation Generator (PCG))**.** *Let C be a reverse-sampleable correlation generator. A pseudorandom correlation generator (PCG) for C is a pair of algorithms* (PCG.Gen, PCG.Expand) *with the following syntax:*

- $(\boldsymbol{s}_0, \boldsymbol{s}_1) \leftarrow \mathsf{PCG.Gen}\left(1^\lambda\right)$*: On input the security parameter λ, it outputs a pair of seeds $(\boldsymbol{s}_0, \boldsymbol{s}_1)$.*
- $R_b \leftarrow \mathsf{PCG.Expand}\left(b, \boldsymbol{s}_b\right)$*: On input an index $b \in \{0,1\}$, the seed $\boldsymbol{s}_b$, it outputs a string R_b.*

Correctness: *We require that the correlation obtained via:*

$$\{(R_0, R_1) \mid (\boldsymbol{s}_0, \boldsymbol{s}_1) \leftarrow \mathsf{PCG.Gen}\left(1^\lambda\right), R_b \leftarrow \mathsf{PCG.Expand}\left(b, \boldsymbol{s}_b\right)\}$$

is indistinguishable from $\{(R_0, R_1) \leftarrow C\left(1^\lambda\right)\}$.

Security: *For any* $b \in \{0,1\}$*, the following two distributions are computationally indistinguishable:*

$$\{(\boldsymbol{s}_{1-b}, R_b) \mid (\boldsymbol{s}_0, \boldsymbol{s}_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda),\ R_b \leftarrow \mathsf{PCG.Expand}\left(b, \boldsymbol{s}_b\right)\},\ \textit{and}$$
$$\left\{(\boldsymbol{s}_{1-b}, R_b) \,\middle|\, \begin{array}{r} (\boldsymbol{s}_0, \boldsymbol{s}_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda), R_{1-b} \leftarrow \mathsf{PCG.Expand}\left(1-b, \boldsymbol{s}_{1-b}\right), \\ R_b \leftarrow \mathsf{Rsample}\left(1-b, R_{1-b}\right) \end{array}\right\}$$

where $\mathsf{Rsample}$ *is the reverse sampling algorithm for correlation* C.

We will also consider m-party PCGs, where $\mathsf{PCG.Gen}(1^\lambda)$ outputs an m-tuple of seeds $(\mathbf{s}_1, \ldots, \mathbf{s}_m)$. Here, security is defined against any subset of colluding parties. In particular, for any $T \subset [m]$, the following two distributions should be computationally indistinguishable:

$$\{(\{\mathbf{s}_j\}_{j\in T}, \{R_i\}_{i\notin T}) \mid (\mathbf{s}_1, \ldots, \mathbf{s}_m) \leftarrow \mathsf{PCG.Gen}(1^\lambda),\ \forall i \notin T, R_i \leftarrow \mathsf{PCG.Expand}\left(i, \mathbf{s}_i\right)\},\ \text{and}$$
$$\left\{(\{\mathbf{s}_j\}_{j\in T}, \{R_i\}_{i\notin T}) \,\middle|\, \begin{array}{r} (\mathbf{s}_1, \ldots, \mathbf{s}_m) \leftarrow \mathsf{PCG.Gen}(1^\lambda), \forall j \in T, R_j \leftarrow \mathsf{PCG.Expand}\left(j, \mathbf{s}_j\right), \\ \{R_i\}_{i\notin T} \leftarrow \mathsf{Rsample}\left(T, \{R_j\}_{j\in T}\right) \end{array}\right\}.$$

3.6.1 PCG for subfield-VOLE

One of the building blocks used in this work is a PCG protocol for subfield-VOLE correlation. It has been studied by the works of [BCG+19b, BCG+19a] and is known to be implied by a suitable choice of the LPN assumption. Our main construction is crucially inspired by such PCG so we give a brief overview of the protocol.

We denote this protocol specifically by $(\mathsf{PCG.Gen}_{\mathsf{sVOLE}}, \mathsf{PCG.Expand}_{\mathsf{sVOLE}})$. Due to the compressing nature of PCG, we also explicitly associate an algorithm sEval with this protocol. It takes as input any compressed vector $y \in \mathbb{F}_p^n$ and an evaluation domain of size $k \le n$, and reconstructs the vector y restricted to $\mathbb{F}_p^k := \mathbb{F}_p^n[:k]$. We denote the compressed form of any vector y by $\widetilde{y}$. Therefore for correctness we always have $y = \mathsf{sEval}\left(\widetilde{y}, n\right)$.

In [BCG+19a], the algorithm $\mathsf{PCG.Gen}_{\mathsf{sVOLE}}$ begins by sampling a random sparse vector $\mathbf{y} \in \mathbb{F}_2^{n'}$ of Hamming weight w and a random offset $\delta \in \mathbb{F}_{2^\lambda}$, but here we alter the syntax so that $\mathsf{PCG.Gen}_{\mathsf{sVOLE}}$ takes these values as input. Since $\mathbf{y}$ is a sparse vector, it can be naturally represented in a compressed form using $O(w \cdot \log(n'))$ bits, which we denote by $\widetilde{\mathbf{y}}$. In this way, $\mathsf{PCG.Gen}_{\mathsf{sVOLE}}$ takes as input $\left(1^\lambda, \widetilde{\mathbf{y}}, \delta\right)$ and outputs a pair of compressed random seeds $(\widetilde{k_0}, \widetilde{k_1})$ where $(\widetilde{k_0}, \widetilde{k_1})$ can later be expanded using sEval into $k_0, k_1 \in \mathbb{F}_{2^\lambda}^{n'}$ such that $k_0 = k_1 + \mathbf{y} \cdot \delta$. In fact, $(\widetilde{k_0}, \widetilde{k_1})$ are the outputs of a Function Secret Sharing (FSS) scheme for the multi-point function induced by the sparse vector $\mathbf{y} \cdot \delta$. Their sizes only depend on $(\lambda, t, \log(n'))$. Furthermore, due to the security of FSS, there exists a simulator Sim so that for any PPT adversary who is given the description of $\mathbf{y} \cdot \delta$, and for each $b \in \{0,1\}$, the two distributions $\left\{(\widetilde{k_b}, \cdot) \leftarrow \mathsf{PCG.Gen}_{\mathsf{sVOLE}}(1^\lambda, \widetilde{\mathbf{y}}, \delta)\right\}$, $\left\{\widetilde{k'_b} \leftarrow \mathsf{Sim}(1^\lambda, \mathbb{F}_{2^\lambda}, n')\right\}$ are indistinguishable.

To expand these seeds into subfield-VOLE correlation, $\mathsf{PCG.Expand}_{\mathsf{sVOLE}}$ takes as input $\left(\widetilde{k_0}, \widetilde{k_1}\right)$ and a random n'-by-n binary code matrix $H_{n',n} \in \mathbb{F}_2^{n' \times n}$ (where $n < n'$), and computes $k_0 = \mathsf{sEval}\left(\widetilde{k_0}, n'\right)$, $k_1 = \mathsf{sEval}\left(\widetilde{k_1}, n'\right)$, $t_0 := k_0 \cdot H_{n',n}$, $t_1 := k_1 \cdot H_{n',n}$ and sets $\mathsf{v} := \mathbf{y} \cdot H_{n',n}$. This immediately gives the desired subfield-VOLE correlation where $t_0 = t_1 + \mathsf{v} \cdot \delta$. The vector v is random due to the LPN assumption.

4 Multiparty NISC with Silent Preprocessing

In this section we describe our first result: a multiparty silent NISC protocol from the LPN assumption in the random oracle model. We organize this section as follows. In section 4.1, we give a definition of multiparty silent NISC. In section 4.2, we revisit the GS18 compiler in the context of multiparty silent NISC and identify a specific type of correlation that we need for implementing this compiler. In section 4.3 we give a PCG protocol for this correlation. The final construction is given in section 4.4. Finally, in section 4.5, we discuss some extensions to our basic protocol. The security proof of our protocol is given in Appendix E.

The main result of this section is the following:

Assuming LPN with inverse polynomial error rate, there exists a multiparty silent NISC protocol in the random oracle model.

4.1 Multiparty Silent NISC: Definition

We introduce the notion of multiparty non-interactive secure computation with silent preprocessing, or *Multiparty Silent NISC*, which extends the two-party silent NISC primitive of [BCG+19a] to the multi-party setting.

An m-party silent NISC protocol begins with a preprocessing phase, where a CRS is sampled and a trusted dealer sets up m secret parameters and distributes them to each party. The computation performed by the dealer should be efficient, in the sense that it only grows with the security parameter, and not with the size of the circuit that the parties will eventually compute. After the preprocessing phase, each party broadcasts a commitment to its input. Finally, the parties compute a circuit C over their joint inputs by broadcasting *one* additional message. Anyone can recover the output of the computation based on these messages.

Definition 4.1 (Multiparty Silent NISC)**.** *An m-party non-interactive secure computation with silent preprocessing (m-party silent NISC) is a protocol described by algorithms* $(\mathsf{Gen}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Compute}, \mathsf{Recover})$ *with the following syntax and properties:*

- $\mathsf{CRS} \leftarrow \mathsf{Gen}(1^\lambda)$*: On input a security parameter λ, the* Gen *algorithm outputs a CRS.*
- $\{s_i\}_{i\in[m]} \leftarrow \mathsf{Setup}\left(1^\lambda, L, \mathsf{CRS}\right)$*: On input the security parameter λ, a bound L on the size of supported circuits, and* CRS*, the Setup algorithm outputs a set of secret parameters $\{s_i\}_{i\in[m]}$. Secret s_i is given to party i.*
- $c_i \leftarrow \mathsf{Commit}\left(i, x_i, s_i, \mathsf{CRS}\right)$*: On input an index i, i^{th} party's input x_i, its secret parameter s_i and* CRS*, the Commit algorithm outputs party i's commitment c_i to its input x_i.*
- $m_i \leftarrow \mathsf{Compute}\left(i, x_i, s_i, \mathsf{CRS}, \{c_j\}_{j\in[m]}, C\right)$*: On input an index i, i^{th} party's input x_i, its secret parameter s_i, the* CRS*, all the commitments $\{c_j\}_{j\in[m]}$ and description of a circuit C, the Compute algorithm outputs party i's message m_i for computing circuit C.*
- $Y \leftarrow \mathsf{Recover}\left(\{m_j\}_{j\in[m]}\right)$*: On input all messages $\{m_j\}_{j\in[m]}$, the Recover algorithm outputs $Y \leftarrow C\left(x_1, \ldots, x_m\right)$.*

4.1.1 Correctness

For any (deterministic) circuit C whose size is bounded by L, and any set of inputs $(x_1, \ldots, x_m)$, correctness requires that:

$$\Pr\left[Y = C\left(x_1, \ldots, x_m\right) \;\middle|\; \begin{array}{r} \mathsf{CRS} \leftarrow \mathsf{Gen}(1^\lambda), \\ \{s_i\}_{i\in[m]} \leftarrow \mathsf{Setup}\left(1^\lambda, L, \mathsf{CRS}\right) \\ c_i \leftarrow \mathsf{Commit}\left(i, x_i, s_i, \mathsf{CRS}\right) \\ m_i \leftarrow \mathsf{Compute}\left(i, x_i, s_i, \mathsf{CRS}, \{c_j\}_{j\in[m]}, C\right) \\ Y \leftarrow \mathsf{Recover}\left(\{m_j\}_{j\in[m]}\right) \end{array} \right] = 1.$$

4.1.2 Silent Preprocessing

A multi-party silent NISC satisfies the following properties.

- *Succinct setup*: The running time of the Setup algorithm is independent of the circuit size L. That is, we require that the setup algorithm runs in some fixed polynomial time $\mathrm{poly}(\lambda)$.
- *Circuit-independent commitment*: The running time of the Commit algorithm is independent of the circuit size, and only depends on the security parameter and input size.

4.1.3 Security

For defining security, we follow the standard real/ideal world paradigm. A formal definition may be found in Appendix C.

4.2 A Strawman From GS18 Compiler

Recall that the GS18 compiler (see Appendix B for a description of the compiler and Appendix A for a description of the *conforming protocol* to which the compiler is applied) yields a two round MPC protocol $(\mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$, which can be presented in the syntax of multiparty NISC as follows:

- The Gen algorithm samples the CRS for GS18 compiler.
- In the setup phase, the setup algorithm samples a set of secret randomness $\{r_i\}_{i\in[m]}$. Then it sets $s_i := r_i$ for each $i \in [m]$.
- In the commit phase, given the description of circuit C, party i commits to its input x_i by running $\left(\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)}\right) \leftarrow \mathsf{MPC}_1\left(1^\lambda, \mathsf{CRS}, C, i, x_i, s_i\right)$. Then it sets $c_i := \mathsf{msg}_i^{(1)}$.
- In the compute phase, given all the previous commitments, i^{th} party computes $\mathsf{msg}_i^{(2)} \leftarrow \mathsf{MPC}_2\left(C,\ \mathsf{st}_i^{(1)}, \{c_j\}_{j\in[m]}\right)$. It then sets $m_i := \mathsf{msg}_i^{(2)}$.
- In the recover phase, given all messages, anyone can simply compute $Y \leftarrow \mathsf{MPC}_3\left(\{m_j\}_{j\in[m]}\right)$.

However, this naive construction does not achieve silent preprocessing (section 4.1.2), due to the fact that MPC_1 takes as input a description of the circuit and that its running time is dependent on the size of this circuit. Thus, this construction does not achieve *circuit-independent commitment*.

To address this issue, we begin by taking a closer look to the MPC_1 algorithm. It outputs two things: an encoding of party i's input, which only depends on the input size, and a number of OT_1 messages that is comparable to the size of circuit C. Merely computing these messages already takes time $O(|C|)$ so we cannot hope to include them as part of the commitment.

The reason these OT_1 messages are required is that, combining with the subsequent OT_2 messages sent by each party's garbled circuits, they allow to set up OT correlations between any two parties (i, j) in the following way. For any round t where party i is the speaking party and party j is one of the listening parties,

- Party i has the receiver strings $R_0 := (\gamma_t, m_{\gamma_t})$. The choice bit γ_t is computed according to the description of action ϕ_t of the conforming protocol[3] Φ: $\gamma_t = \mathsf{NAND}(\mathsf{v}_f \oplus \alpha, \mathsf{v}_g \oplus \beta) \oplus \mathsf{v}_h$, where α and β are recorded in each party's public state, $\phi_t := (i, f, g, h)$ and $\mathsf{v} := \mathsf{v}_i$ are i^{th} party masking bits (secret state).
- Party j has the sender strings $R_1 := \left(m_0 := \mathbf{lab}_{h,0}^{i,t+1}, m_1 := \mathbf{lab}_{h,1}^{i,t+1}\right)$, where $\left(\mathbf{lab}_{h,0}^{i,t+1}, \mathbf{lab}_{h,1}^{i,t+1}\right)$ are input labels for input value γ_t of its next garbled circuit.

Then party i can simply output its string so that any party can recover the correct input label (c.f $\mathbf{lab}_{h,\gamma_t}^{i,t+1}$) for party j's next round garbled circuit.

We formalize those OT correlations by defining the more general GS18 correlations:

Definition 4.2 (GS18 correlation generator)**.** *A GS18 correlation generator, denoted as C_{GS}, is an algorithm which takes as input the security parameter λ and a set $\{\phi_t := (\cdot, f, g, h)\}_{t\in[q]}$, and outputs:*

$$\begin{pmatrix} R_0 := \left(\mathsf{v}, \left\{m_{t,(\alpha,\beta)}\right\}_{\alpha,\beta\in\{0,1\},t\in[q]}\right), \\ R_1 := \left(\delta, \left\{m_{t,(\alpha,\beta),0}\right\}_{\alpha,\beta\in\{0,1\},t\in[q]}\right) \end{pmatrix} \leftarrow C_{\mathsf{GS}}\left(1^\lambda, \{\phi_t\}_{t\in[q]}\right)$$

Where $\mathsf{v} \leftarrow \{0,1\}^n$ is a random vector and $\delta \leftarrow \mathbb{F}_{2^\lambda}$ is a random offset. $m_{t,(\alpha,\beta),0} \leftarrow \{0,1\}^\lambda$ is a random string. Furthermore, for each $t \in [q]$, $\phi_t := (\cdot, f, g, h)$, and for any choice of $\alpha, \beta \in \{0,1\}$, let $\gamma_{t,(\alpha,\beta)} = \mathsf{NAND}(\mathsf{v}_f \oplus \alpha, \mathsf{v}_g \oplus \beta) \oplus \mathsf{v}_h$, and $m_{t,(\alpha,\beta),1} := m_{t,(\alpha,\beta),0} + \delta$. Then $m_{t,(\alpha,\beta)} = m_{t,(\alpha,\beta),\gamma_{t,(\alpha,\beta)}}$. Notice that the GS18 correlation is also reverse-sampleable:

- $\mathsf{Rsample}(0, R_0)$*: Sample $\delta \leftarrow \mathbb{F}_{2^\lambda}$ randomly, then for each $t \in [q]$ and each $\alpha, \beta \in \{0,1\}$, set $m_{t,(\alpha,\beta),0} := m_{t,(\alpha,\beta)} + \gamma_{t,(\alpha,\beta)} \cdot \delta$.*
- $\mathsf{Rsample}(1, R_1)$*: Sample $\mathsf{v} \leftarrow \{0,1\}^n$ randomly, then for each $t \in [q]$ and each $\alpha, \beta \in \{0,1\}$, compute $\gamma_{t,(\alpha,\beta)}$ as before and set $m_{t,(\alpha,\beta)} := m_{t,(\alpha,\beta),0} + \gamma_{t,(\alpha,\beta)} \cdot \delta$.*

Observe that we define GS18 correlation such that for each action ϕ_t, we obtain a set of four correlated OTs, one for each choice of α, β:

$$R_0 := \left(\gamma_{t,(\alpha,\beta)}, m_{t,(\alpha,\beta)}\right),\ R_1 := \left(m_{t,(\alpha,\beta),0}, m_{t,(\alpha,\beta),1} := m_{t,(\alpha,\beta),0} + \delta\right) \tag{1}$$

As first observed in [IKNP03], it suffices to use a correlation robust hash function to obtain random OTs from correlated OTs. In our construction we deploy a random oracle function ρ as correlation robust hash function.

Now suppose that before the compute phase, for each round t, party i is given R_0 whereas party j is given R_1, and additionally both parties agree on the choice of (α, β) in the compute phase. Thereby party i's garbled circuit can simply output $\left(\gamma_t := \gamma_{t,(\alpha,\beta)}^i, m_t := \rho\left(m_{t,(\alpha,\beta)}^{i,j}\right)\right)$, whereas party j's garbled circuit outputs:

$\left(a_0 := \mathbf{lab}_{h,0}^{i,t+1} \oplus \rho\left(m_{t,(\alpha,\beta),0}^{i,j}\right), a_1 := \mathbf{lab}_{h,1}^{i,t+1} \oplus \rho\left(m_{t,(\alpha,\beta),1}^{i,j}\right)\right)$. As a result, any party can recover the label $\mathbf{lab}_{h,\gamma_t}^{i,t+1} = a_{\gamma_t} \oplus m_t$.

But how can those parties obtain GS18 correlations before the compute phase? We cannot afford to generate them in the setup phase since its runtime should be succinct. To solve this problem, we specifically design a pseudorandom correlation generator (PCG) for GS18 correlations. With this tweak, the setup algorithm will include PCG seeds for each party in its secret parameter, so that each party can silently expand its seed to obtain desired GS18 correlations before the compute phase, hence making the preprocessing phase silent.

[3]See Appendix A for a description of the conforming protocol.

4.3 A PCG Protocol For GS18 Correlation

As suggested in [BCG+19a], any subfield-VOLE correlation gives correlated OTs where for each $i \in [n]$, the receiver string is $R_0 := \left(\mathsf{v}[i], m_{\mathsf{v}[i]} := t_0[i]\right)$, and the sender string is $R_1 := (m_0 := t_1[i], m_1 := t_1[i] + \delta)$. One can then get random OTs by applying a correlation-robust hash function on these correlated OTs.

In order to generate desired OT correlations, first note that in the field $\mathbb{F}_2$, one can rewrite each NAND relation as a degree-two equation: $\mathsf{NAND}\left(\mathsf{v}_f \oplus \alpha, \mathsf{v}_g \oplus \beta\right) \oplus \mathsf{v}_h \equiv 1 + \left(\mathsf{v}_f + \alpha\right)\left(\mathsf{v}_g + \beta\right) + \mathsf{v}_h = \left(\mathsf{v}_f \mathsf{v}_g\right) + \left(\alpha \mathsf{v}_g + \beta \mathsf{v}_f + \mathsf{v}_h\right) + (\alpha\beta + 1)$. As a result of this, given random masking bits $\mathsf{v} \in \{0,1\}^n$, each choice bit γ_t can be viewed as a sum of a degree two relation over v, a degree one relation over v, and a constant which are parametrized by the choice of (α, β).

The subfield-VOLE correlation is itself a degree 1 relation. As before we set $\mathsf{v} := \mathbf{y} \cdot H_{n',n}$. In order to distinguish it from a degree 2 relation, we use the notation $((\widetilde{\mathbf{y}}, \widetilde{k_0^1}),\ (\widetilde{k_1^1}, \delta))$ to denote the degree 1 seeds for receiver $(R := 0)$ and sender $(S := 1)$ respectively, and propagate this notation to all other symbols in the natural way. For consistency with previous sections, we slightly abuse the notation by letting $\mathsf{v}_i := \mathsf{v}[i]$. Under this notation, the degree-1 correlated OT can be rewritten as follows: for each $i \in [n]$, $R_0^1 := \left(\mathsf{v}_i,\ m_{\mathsf{v}_i}^1 := t_0^1[i]\right)$, $R_1^1 := \left(m_0^1 := t_1^1[i], m_1^1 := t_1^1[i] + \delta\right)$.

In order to deduce degree 2 relations, we take the tensor product of same error vector with itself and use it as the new error vector as suggested in [BCG+19a]: $\left(\widetilde{k_0^2}, \widetilde{k_1^2}\right) \leftarrow \mathsf{PCG.Gen}_{\mathsf{sVOLE}}\left(1^\lambda, \widetilde{\mathbf{y} \otimes \mathbf{y}}, \delta\right)$. The expansion algorithm also needs to be modified as follows: $k_0^2 = \mathsf{sEval}\left(\widetilde{k_0^2}, n'\right)$, $t_0^2 := k_0^2 \cdot \left(H_{n',n} \otimes H_{n',n}\right), k_1^2 = \mathsf{sEval}\left(\widetilde{k_1^2}, n'\right)$, $t_1^2 := k_1^2 \cdot (H_{n',n} \otimes H_{n',n})$, where $t_0^2, t_1^2 \in \mathbb{F}_{2^\lambda}^{n^2}$. Viewing both t_1^2 and t_0^2 as n-by-n matrices over $\mathbb{F}_{2^\lambda}$, for any $i, j \in [n]$, observe that the following degree 2 relation holds: $t_0^2[i,j] = t_1^2[i,j] + \mathsf{v}_i \mathsf{v}_j \cdot \delta$. As before, this immediately gives a correlated OT where $R_0^2 := \left(\mathsf{v}_i \mathsf{v}_j, m_{\mathsf{v}_i \mathsf{v}_j}^2 := t_0^2[i,j]\right)$, and $R_1^2 := \left(m_0^2 := t_1^2[i,j], m_1^2 := t_1^2[i,j] + \delta\right)$.

Now that we know how to generate degree 1 and degree 2 correlated OTs, we can easily derive the GS18 correlations by taking linear combinations of (R_0^1, R_0^2) (resp. (R_1^1, R_1^2)) over $\mathbb{F}_2$. This gives a PCG protocol for generating GS18 correlations. Now, the protocol we need is actually in the *multi-party* setting: that is, the receiver's choice bits v must be shared between all of their pairwise correlations with every other sender. This additional requirement can be ensured by reusing the same error vector $\mathbf{y}$ multiple times. Below we give a PCG protocol for GS18 correlations with one receiver and an arbitrary number m of senders. We denote this specific PCG protocol by $(\mathsf{PCG.Gen}_{\mathsf{GS}}, \mathsf{PCG.Expand}_{\mathsf{GS}})$, given in Protocol 1.

We prove the following theorem in Appendix D.

Theorem 4.3. *Assuming LPN with inverse polynomial noise rate,* $(\mathsf{PCG.Gen}_{\mathsf{GS}}, \mathsf{PCG.Expand}_{\mathsf{GS}})$ *in Protocol 1 is a multi-party PCG protocol for GS18 correlations satisfying PCG security (Definition 3.11).*

Protocol 1 (Multi-party PCG Protocol For GS18 Correlations)**.**

- *Parameters: Let λ be the security parameter, m be the number of senders, q be the number of actions for the GS18 protocol, and n', n be integers such that $n' > n$.*
- *Output:*
  - ***For receiver**:*
    - *Masking bits $\mathsf{v} \in \{0,1\}^n$;*
    - *For each $t \in [q]$, $\alpha, \beta \in \{0,1\}$, $i \in [m]$, a receiver string $m^i_{t,(\alpha,\beta)}$.*
  - ***For sender** $i \in [m]$:*
    - *A shift $\delta_i \in \mathbb{F}_{2^\lambda}$;*
    - *For each $t \in [q]$, $\alpha, \beta \in \{0,1\}$, a sender string $m^i_{t,(\alpha,\beta),0}$.*
- *Input:*
  - *A compressed random error vector $\mathbf{y} \in \{0,1\}^{n'}$ with hamming weight λ, denoted by $\widetilde{\mathbf{y}}$.*
  - *A random shift $\delta_i \in \mathbb{F}_{2^\lambda}$ for each $i \in [m]$.*
  - *An n'-by- n binary code matrix $H_{n',n}$.*
  - *A sequence of actions $\{\phi_t\}_{t\in[q]}$.*
- $\mathsf{Gen}_{\mathsf{GS}}\left(1^\lambda, \widetilde{\mathbf{y}}, \{\delta_i\}_{i\in[m]}\right)$*:*
  - *For each $i \in [m]$, compute* $\left(\widetilde{k^1_{i,0}}, \widetilde{k^1_{i,1}}\right) \leftarrow \mathsf{PCG.Gen}_{\mathsf{sVOLE}}\left(1^\lambda, \widetilde{\mathbf{y}}, \delta_i\right)$; $\left(\widetilde{k^2_{i,0}}, \widetilde{k^2_{i,1}}\right) \leftarrow \mathsf{PCG.Gen}_{\mathsf{sVOLE}}\left(1^\lambda, \widetilde{\mathbf{y} \otimes \mathbf{y}}, \delta_i\right)$.
  - *Set* $\boldsymbol{s}^i_0 := \left(\widetilde{k^1_{i,0}}, \widetilde{k^2_{i,0}}, \widetilde{\mathbf{y}}\right)$, $\boldsymbol{s}^i_1 := \left(\widetilde{k^1_{i,1}}, \widetilde{k^2_{i,1}}, \delta_i\right)$.
- $\mathsf{Expand}_{\mathsf{GS}}\left(1^\lambda, b, \{\boldsymbol{s}^i_b\}_{i\in[m]}, H_{n',n}, \{\phi_t\}_{t\in[q]}\right)$:
  - *If $b = 0$, set $\mathbf{y} = \mathsf{sEval}(\widetilde{\mathbf{y}}, n')$, $\mathsf{v} = \mathbf{y} \cdot H_{n',n}$, and for each $i \in [m]$:*
    - *Parse $\boldsymbol{s}^i_0 = \left(k^1_{i,0}, k^2_{i,0}, \widetilde{\mathbf{y}}\right)$, and compute $k^1_{i,0} = \mathsf{sEval}\left(\widetilde{k^1_{i,0}}, n'\right)$, $k^2_{i,0} = \mathsf{sEval}\left(\widetilde{k^2_{i,0}}, n'\right)$.*
    - *Compute $t^1_{i,0} := k^1_{i,0} \cdot H_{n',n}$, $t^2_{i,0} := k^2_{i,0} \cdot (H_{n',n} \otimes H_{n',n})$.*
    - *For each $t \in [q]$, parse $\phi_t := (\cdot, f, g, h)$, and for each $\alpha, \beta \in \{0,1\}$ set $m^i_{t,(\alpha,\beta)} := t^2_{i,0}[f,g] + \alpha \cdot t^1_{i,0}[g] + \beta \cdot t^1_{i,0}[f] + t^1_{i,0}[h]$.*
  - *If $b = 1$, for each $i \in [m]$:*
    - *Parse $\boldsymbol{s}^i_1 = \left(k^1_{i,1}, k^2_{i,1}, \delta_i\right)$, and compute $k^1_{i,1} = \mathsf{sEval}\left(\widetilde{k^1_{i,1}}, n'\right)$, $k^2_{i,1} = \mathsf{sEval}\left(\widetilde{k^2_{i,1}}, n'\right)$.*
    - *Compute $t^1_{i,1} := k^1_{i,1} \cdot H_{n',n}$, $t^2_{i,1} := k^2_{i,1} \cdot (H_{n',n} \otimes H_{n',n})$.*
    - *For each $t \in [q]$, parse $\phi_t := (\cdot, f, g, h)$, and for each $\alpha, \beta \in \{0,1\}$, set $m^i_{t,(\alpha,\beta),0} := t^2_{i,1}[f,g] + \alpha \cdot t^1_{i,1}[g] + \beta \cdot t^1_{i,1}[f] + t^1_{i,1}[h] + (\alpha\beta + 1) \cdot \delta_i$.*

4.4 Multiparty Silent NISC: The Construction

4.4.1 Two-step seed expansion

In our strawman protocol (see section 4.2), each party's commitment contains an encoding of its input and a large number of OT_1 messages. Using PCG for GS18 correlations we are able to remove the OT_1 messages in this commitment. Nevertheless, recall that in GS18 compiler, party i's input encoding is computed as $z_i := x_i \oplus r_i$, where $r_i := \mathsf{v}_i[: l]$ (l is a bound on $|x_i|$). If party i does this naively and computes the whole masking bits v_i in the commit phase, it would take time $|\mathsf{v}_i|$ at least, which is dependent on the circuit size. To circumvent this problem, we slightly modify the receiver expansion algorithm to allow a two-step seed expansion.

First, instead of generating the code matrix $H_{n',n}$ uniformly at random, we let $H_{n',n}$ be a block diagonal matrix that consists of a small matrix $H^1_{l',l}$ and a big matrix $H^2_{n'-l',n-l}$ along its diagonal. The small matrix is only used to generate input masking bits whereas the big matrix is used to generate all of the remaining masking bits. Correspondingly, we also need to modify the error vector $\mathbf{y}$ now that $H_{n',n}$ is not a uniformly random matrix. The error vector will be split into two parts: $\mathbf{y} := \mathbf{y}'||\mathbf{y}^*$, where $|\mathbf{y}'| = l'$ and $|\mathbf{y}^*| = n' - l'$. We sample $\mathbf{y}' \leftarrow \mathcal{HW}_{l',\lambda}$ and $\mathbf{y}^* \leftarrow \mathcal{HW}_{n'-l',\lambda}$ independently. This ensures that both $\mathsf{v}' = \mathbf{y}' \cdot H^1_{l',l}$ and $\mathsf{v}^* = \mathbf{y}^* \cdot H^2_{n'-l',n-l}$ will both be indistinguishable from random due to the LPN assumption with inverse polynomial noise rate, showing that the multi-party PCG from last section remains secure.

Then, in the input commitment phase, each party computes $\mathbf{y}' = \mathsf{sEval}(\widetilde{\mathbf{y}'}, l')$ and then sets $\mathsf{v}' = \mathbf{y}' \cdot H^1_{l',l}$ and $c_i := z_i = x_i \oplus \mathsf{v}'$. This can be seen as the first-step seed expansion and it allows to remove dependency on circuit size. Finally, in the compute phase, each pair of parties silently expand the rest seeds just as before. This is the second-step seed expansion.

Protocol 2 (Multiparty Silent NISC)**.**

- *Parameters: Let m be the number of parties. Let $(\mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$ be a set of algorithms in the GS18 compiler, and let $(\mathsf{PCG.Gen}_{\mathsf{GS}}, \mathsf{PCG.Expand}_{\mathsf{GS}})$ be a multi-party PCG protocol for GS18 correlations. Let $n' > n$ be integers that depend on the size L of the circuit to be computed, and let $l' > l$ be integers that depends on the size of inputs to the circuit.*
- $\mathsf{Gen}(1^\lambda)$*: Set $\mathsf{CRS} := \rho$, where ρ is a random oracle function.*
- $\mathsf{Setup}(1^\lambda, L)$*:*
  1. *For each $i \in [m]$, sample $\mathbf{y}'_i \leftarrow \mathcal{HW}_{l',\lambda}$, $\mathbf{y}^*_i \leftarrow \mathcal{HW}_{n'-l',\lambda}$ and set $\mathbf{y}_i = \mathbf{y}'_i||\mathbf{y}^*_i$.*
  2. *For each $i, j \in [m]$, sample shifts $\delta_{i,j} \leftarrow \mathbb{F}_p$.*
  3. *For each $i \in [m]$, compute $\left\{\boldsymbol{s}_0^{i,j}, \boldsymbol{s}_1^{i,j}\right\}_{j \neq i} \leftarrow \mathsf{PCG.Gen}_{\mathsf{GS}}\left(1^\lambda, \widetilde{\mathbf{y}}_i, \{\delta_{i,j}\}_{j\neq i}\right)$.*
  4. *Set secret parameter $s_i := \left(\left\{\boldsymbol{s}_0^{i,j}\right\}_{j\in[m]/\{i\}}, \left\{\boldsymbol{s}_1^{j,i}\right\}_{j\in[m]/\{i\}}\right)$.*
- $\mathsf{Commit}\,(i, x_i, s_i, \mathsf{CRS})$*:*
  1. *Parse $s_i := \left(\left\{\boldsymbol{s}_0^{i,j}\right\}_{j\in[m]/\{i\}}, \left\{\boldsymbol{s}_1^{j,i}\right\}_{j\in[m]/\{i\}}\right)$, then parse any $\boldsymbol{s}_0^{i,j} := \left(\widetilde{k}_0^1, \widetilde{k}_0^2, \widetilde{\mathbf{y}}_i\right)$ and $\widetilde{\mathbf{y}}_i = \widetilde{\mathbf{y}'_i}||\widetilde{\mathbf{y}^*_i}$.*
  2. *Compute ${\mathbf{y}_i}' = \mathsf{sEval}\left(\widetilde{\mathbf{y}'_i}, l'\right)$.*
  3. *Generate an l'-by- l random binary code matrix $H^1_{l',l} \leftarrow \rho\,(l, l')$ and compute ${\mathsf{v}_i}' = {\mathbf{y}_i}' \cdot H^1_{l',l}$, where $l \geq |x_i|$ and $l' > l$.*
  4. *Set commitment $c_i := x_i \oplus {\mathsf{v}_i}'$.*
- $\mathsf{Compute}$*: See algorithm* 4.4.
- $\mathsf{Recover}$*: See algorithm* 4.5.

Algorithm 4.4 (Compute).

- *Parameters: Let C be the description of a circuit and Φ be a T-round conforming protocol for computing C.*
- $\mathsf{Compute}\left(i, x_i, s_i, \mathsf{CRS}, \{c_j\}_{j\in[m]}, C\right)$:

  1. *Parse* $s_i := \left(\{\boldsymbol{s}_0^{i,j}\}_{j\in[m]/\{i\}}, \{\boldsymbol{s}_1^{j,i}\}_{j\in[m]/\{i\}}\right)$.
  2. *Generate a* $(n'-l')$*-by-*$(n-l)$ *random binary code matrix* $H^2_{n'-l',n-l} \leftarrow \rho(n',l')$*, and set*

  $$H_{n',n} := \begin{bmatrix} H^1_{l',l} & \\ & H^2_{n'-l',n-l} \end{bmatrix}$$

  3. $\left(\mathsf{v}_i, \left\{m^{i,j}_{t,(\alpha,\beta)}\right\}_{t,\alpha,\beta,j}\right) \leftarrow \mathsf{PCG.Expand}_{\mathsf{GS}}\left(1^\lambda, 0, \{\boldsymbol{s}_0^{i,j}\}_{j\neq i}, H_{n',n}, \{\phi_t\}_{t\in[q]}\right)$, $\left(\delta_{j,i}, \left\{m^{j,i}_{t,(\alpha,\beta),0}\right\}_{t,\alpha,\beta,j}\right) \leftarrow \mathsf{PCG.Expand}_{\mathsf{GS}}\left(1^\lambda, 1, \{\boldsymbol{s}_1^{j,i}\}_{j\neq i}, H_{n',n}, \{\phi_t\}_{t\in[q]}\right)$.
  4. *For each* $t \in T$ *such that* $\phi_t := (i,f,g,h)$*, and each* $j \in [m]/\{i\}$*, compute* $\left\{\widetilde{m}^{i,j}_{t,(\alpha,\beta)}\right\}_{t,\alpha,\beta} := \left\{\rho\left(m^{i,j}_{t,(\alpha,\beta)}\right)\right\}_{t,\alpha,\beta}$.

     For each $t \in T$ *such that* $\phi_t := (j,f,g,h)$ *for* $j \neq i$,
     - *Set* $\left\{m^{j,i}_{t,(\alpha,\beta),1}\right\}_{t,\alpha,\beta} := \left\{m^{j,i}_{t,(\alpha,\beta),0} + \delta_{j,i}\right\}_{t,\alpha,\beta}$.
     - $\left\{\widetilde{m}^{j,i}_{(t,(\alpha,\beta),0)}, \widetilde{m}^{j,i}_{(t,(\alpha,\beta),1)}\right\}_{t,\alpha,\beta} := \left\{\rho\left(m^{j,i}_{(t,(\alpha,\beta),0)}\right), \rho\left(m^{j,i}_{(t,(\alpha,\beta),1)}\right)\right\}_{t,\alpha,\beta}$
  5. *Parse* $\mathsf{v}_i := \mathsf{v}_i{}'||\mathsf{v}_i^*$ *and adjust* v_i^* *so that* $pq = |\mathsf{v}_i^*|$*. Initialize a computation tape* $\mathsf{st}_i := c_1||0^{pq}||\ldots||c_m||0^{pq}$*. Let* $N := |\mathsf{st}_i|$.
  6. *Set* $\overline{\boldsymbol{lab}}^{i,T+1} := \left(\boldsymbol{lab}^{i,T+1}_{k,0}, \boldsymbol{lab}^{i,T+1}_{k,1}\right)_{k\in[N]}$ *where for each* $k \in [N]$ *and* $b \in \{0,1\}$ $\boldsymbol{lab}^{i,T+1}_{k,b} = 0^\lambda$.
  7. *For each* t *from* T *to 1, compute:*

  $$\left(\widetilde{\mathsf{P}}^{i,t}, \overline{\boldsymbol{lab}}^{i,t}\right) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{P}^{i,t}\right).$$

     where the circuit $\mathsf{P}^{i,t}$ *hardcodes party* i*'s receiver and sender strings, as well as all input labels of* $\mathsf{P}^{i,t+1}$ *(see algorithm 4.6).*
  8. *Set* $\widehat{\boldsymbol{lab}}^{i,1} := \left\{\boldsymbol{lab}^{i,1}_{k,\mathsf{st}_{i,k}}\right\}_{k\in[N]}$
  9. *Set message* $m_i := \left(\left\{\widetilde{\mathsf{P}}^{i,t}\right\}_{t\in[T]}, \widehat{\boldsymbol{lab}}^{i,1}\right)$.

Algorithm 4.5 (Recover)**.**

- *Parameters: Let Φ be the conforming protocol that computes circuit C. Let T be total number of rounds of Φ.*
- $\mathsf{Recover}\left(\{m_j\}_{j\in[m]}\right)$*:*

  1. *For each $j \in [m]$, parse $m_j := \left(\left\{\widetilde{\mathsf{P}}^{j,t}\right\}_{t\in[T]}, \widehat{\boldsymbol{lab}}^{j,1}\right)$.*
  2. *For each t from 1 to T, do:*
     (a) *Parse action $\phi_t := (i^*, f, g, h)$.*
     (b) *Compute $\left(\gamma_t, \left\{\widetilde{m}_t^{i^*,j}\right\}_{j\in[m]/\{i^*\}}, \widehat{\boldsymbol{lab}}^{i^*,t+1}\right) \leftarrow \mathsf{GEval}\left(\widetilde{\mathsf{P}}^{i^*,t}, \widehat{\boldsymbol{lab}}^{i^*,t}\right)$.*
     (c) *For each $j \neq i^*$, do:*
        i. *Compute $(a_0,\ a_1) \leftarrow \mathsf{GEval}\left(\widetilde{\mathsf{P}}^{j,t}, \widehat{\boldsymbol{lab}}^{j,t}\right)$.*
        ii. *Recover $\boldsymbol{lab}_h^{j,t+1} = a_{\gamma_t} \oplus \widetilde{m}_t^{i^*,j}$.*
        iii. *Reset $\widehat{\boldsymbol{lab}}^{j,t+1} := \left\{\{\boldsymbol{lab}_{k,\mathsf{st}_{j,k}}^{j,t+1}\}_{k\in[N]/\{h\}}, \boldsymbol{lab}_h^{j,t+1}\right\}$.*
  3. *Let $\mathsf{Z} := (\gamma_1, \ldots, \gamma_T)$, set $Y := \mathsf{post}(\mathsf{Z})$.*

Algorithm 4.6 (Circuit $\mathsf{P}^{i,t}$)**.**
Input: st_i.
Hardwired inputs: Party i's masking bits v_i, its receiver and sender strings $\left\{\widetilde{m}_{t,(\alpha,\beta)}^{i,j}\right\}_{\alpha,\beta\in\{0,1\},j\in[m]/\{i\}}$, $\left\{\left(\widetilde{m}_{(t,(\alpha,\beta),0)}^{j,i}, \widetilde{m}_{(t,(\alpha,\beta),1)}^{j,i}\right)\right\}_{\alpha,\beta\in\{0,1\},j\in[m]/\{i\}}$, the input labels of the next garbled circuit $\widetilde{\mathsf{P}}^{i,t+1}$: $\overline{\boldsymbol{lab}}^{i,t+1}$, and the round action ϕ_t.

1. *Parse $\phi_t = (i^*, f, g, h)$.*
2. *Set $\alpha := \mathsf{st}_i[(i^*-1)(pq+l)+f]$, $\beta := \mathsf{st}_i[(i^*-1)(pq+l)+g]$.*
3. *If $i = i^*$, then:*
   (a) *Set $\mathsf{v} := \mathsf{v}_i$, and compute $\gamma_{t,(\alpha,\beta)}^i = \mathit{NAND}(\mathsf{v}_f \oplus \alpha, \mathsf{v}_g \oplus \beta) \oplus \mathsf{v}_h$.*
   (b) *Set $\mathsf{st}_i[(i-1)(pq+l)+h] := \gamma_{t,(\alpha,\beta)}^i$.*
   (c) *Set $\widehat{\boldsymbol{lab}}^{i,t+1} := \left\{\boldsymbol{lab}_{k,\mathsf{st}_{i,k}}^{i,t+1}\right\}_{k\in[N]}$.*
   (d) *Output $\left(\gamma_{t,(\alpha,\beta)}^i,\ \left\{\widetilde{m}_{t,(\alpha,\beta)}^{i,j}\right\}_{j\in[m]/\{i\}},\ \widehat{\boldsymbol{lab}}^{i,t+1}\right)$.*
4. *If $i \neq i^*$, then:*
   (a) *Set $\widehat{\boldsymbol{lab}}^{i,t+1} := \left\{\boldsymbol{lab}_{k,\mathsf{st}_{i,k}}^{i,t+1}\right\}_{k\in[N]/\{h\}}$.*
   (b) *Output $\left(\boldsymbol{lab}_{h,0}^{i,t+1} \oplus \widetilde{m}_{t,(\alpha,\beta),0}^{i^*,i},\ \boldsymbol{lab}_{h,1}^{i,t+1} \oplus \widetilde{m}_{t,(\alpha,\beta),1}^{i^*,i},\ \widehat{\boldsymbol{lab}}^{i,t+1}\right)$, where the label $\boldsymbol{lab}_h^{i,t+1}$ is the input for the bit $\mathsf{st}_i[(i^*-1)(pq+l)+h]$ of the next garbled circuit.*

Theorem 4.7. *Assuming LPN with inverse polynomial error rate, Protocol 2 is a secure multiparty silent NISC protocol in the random oracle model.*

See Section 3.5 for how we set LPN parameters based on the (polynomial-size) circuit C to be computed. The proof of this theorem is given in Appendix E.

4.5 Extensions

4.5.1 Removing the random oracle

Our construction of multiparty silent NISC relies on a random oracle. Nevertheless, we can remove the use of random oracle, at the cost of introducing a large (growing with the size of the computation) CRS. Below we define this notion as multiparty silent NISC with large reusable CRS.

To begin with, one can observe that the previous construction utilizes the random oracle in two following ways:

- Modeling it as a correlation robust hash function; This is used to obtain random OTs from correlated OTs.
- Generating random binary code matrices for PCG seed expansion.

As already observed in [BCG$^+$19b], the role of correlation robust hash function can be replaced by an encryption scheme which is semantically secure against related-key attack for the class of linear functions. It was also shown that this encryption scheme can be based on standard LPN assumptions (over $\mathbb{F}_2$)[AHI11]. Therefore we can effectively remove this use of random oracle without introducing new assumptions.

Without using the random oracle, an easy way to solve the second problem is to let the Gen algorithm sample a random block-diagonal code matrix, and directly includes it in the CRS. This, however, requires that the Gen algorithm must take as input the circuit size bound L since the dimension of this code matrix must exceed the size of circuit to be computed. Furthermore, the CRS is large since its size now depends on the circuit size. As a result, the commit algorithm cannot take the whole CRS as input. So instead we split the block-diagonal code matrix, and only supply the small code matrix as input to the commit algorithm so as to remove its dependency on the circuit size. To summarize, we set $\mathsf{CRS} := (\mathsf{CRS}', \mathsf{CRS}^*) \leftarrow \mathsf{Gen}(1^\lambda, L)$ where $\mathsf{CRS}' := H^1_{l',l}$ and $\mathsf{CRS}^* := H^2_{n'-l',n-l}$. Notice that the size of CRS' only depends on the input size whereas CRS^* depends on the circuit size $|C| \leq L$. The commit algorithm now takes as input $(i, x_i, s_i, \mathsf{CRS}')$ whereas the compute algorithm still takes as input $\left(i, x_i, s_i, \mathsf{CRS}, \{c_j\}_{j\in[m]}, C\right)$. We adopt these notations for CRS in all following sections.

4.5.2 Reusable CRS

Although the CRS in the resulting protocol is large, it can be *reused* across an arbitrary polynomial number of multiparty silent NISC executions. This property will be crucial for our construction next section, so we give more details here. After the CRS is sampled, an adversary may specify any polynomial $q(\lambda)$ number of multiparty silent NISC executions in which it would like to participate using the same fixed CRS (but fresh preprocessing, commitment, and compute phases). Security for each of these executions will still follow from the LPN assumption. To see why, recall that the CRS is a dual-LPN matrix H, and is only used in the security proof when appealing to the dual-LPN assumption. By a straightforward hybrid argument, dual-LPN will hold with respect to a single random matrix H for any polynomial $q(\lambda)$ number of samples.

5 Reusable Two-Round MPC from LPN

In this section, we build on top of our previous result and show a compiler that takes any multiparty silent NISC with large reusable CRS and produces a reusable two-round MPC protocol. This section is organized as follows: we divide our compiler into three parts, each part involving one specific transformation. We proceed and give constructions of these transformations one by one in each subsection:

1. We define the notion of bounded FMS-MPC and show that multiparty silent NISC with reusable large CRS implies bounded FMS-MPC.
2. We show that bounded FMS-MPC implies standard FMS-MPC
3. Finally, we appeal to [BGMM20], who show that FMS-MPC implies reusable two-round MPC.

5.1 Multiparty silent NISC with reusable large CRS $\rightarrow$ Bounded FMS-MPC

We start by defining a relaxed notion of first message succinct MPC (FMS-MPC), which was introduced in [BGMM20]. We call this new primitive a bounded FMS-MPC, which can be naturally thought as a middle ground between a multiparty silent NISC and a standard FMS-MPC.

Definition 5.1 (Bounded FMS-MPC)**.** *Let* Gen *be an algorithm that generates a* CRS*. We say that the protocol* $\pi^* = (\mathsf{Gen}, \mathsf{BFMS.MPC}_1, \mathsf{BFMS.MPC}_2, \mathsf{BFMS.MPC}_3)$ *is a bounded FMS-MPC protocol if it is a two-round MPC protocol with the following properties:*

- *Bounded circuit size: The* Gen *algorithm takes as input the security parameter* λ*, a circuit size bound* L*, and outputs a* $\mathsf{CRS} := (\mathsf{CRS}', \mathsf{CRS}^*)$*. The size of* CRS' *only depends on an upper bound on input size, whereas the size of* CRS^* *can be as large as* L*. Moreover, the protocol* π^* *only supports circuits such that* $|C| \leq L$*.*
- *Reusable CRS: The part* CRS^* *only needs to be set up once, and can be reused across an unbounded polynomial number of two-round MPC protocols.*
- *First message succinctness: The* $\mathsf{BFMS.MPC}_1$ *algorithm takes as input* $\left(1^\lambda, \mathsf{CRS}', i, x_i\right)$*. In particular, its runtime should not depend on the circuit size* $|C|$*.*

As our construction of bounded FMS-MPC from multiparty silent NISC is very similar to the transformation given in [BGMM20, Section 5], we defer the construction and security proof to Appendix F. In particular, we prove the following theorem.

Theorem 5.2. *Assuming a semi-honest multiparty silent NISC with large reusable CRS and a maliciously-secure vanilla two-round MPC in the CRS model, there exists a maliciously-secure bounded FMS-MPC protocol.*

Due to results from last section and [DGH+20], we have the following corollary.

Corollary 5.3. *Assuming LPN with inverse polynomial error rate, there exists a maliciously-secure bounded FMS-MPC protocol.*

5.2 Bounded FMS-MPC $\rightarrow$ FMS-MPC

In order to obtain standard FMS-MPC, we must allow for computation of a priori unbounded polynomial size circuits. That is, we must support the computation of unbounded polynomial size circuits using only a bounded polynomial size CRS. A natural idea is then to use *randomized encodings* to break down the computation of any unbounded polynomial size circuit into the computation of a number of *bounded* polynomial size circuits, and use a bounded size (reusable) CRS to compute each small circuit.

Indeed, any m-input polynomial-size circuit $C : \{0,1\}^{m\cdot\ell} \rightarrow \{0,1\}^{\ell'}$ admits a randomized encoding (see section 3.4), which can be written as a sequence of small circuits $\{G_y : \{0,1\}^{m\cdot\ell} \times \{0,1\}^\lambda \rightarrow \{0,1\}^s\}_{y\in[n]}$, where n depends on the size of C, but each G_y has size $p(\lambda)$ for some a priori *fixed* polynomial $p(\cdot)$. The correctness of randomized encoding ensures that for any inputs $x_1, \ldots, x_m$ and random coins $v \leftarrow \{0,1\}^\lambda$, one can recover the output $Y := C(x_1, \ldots, x_m)$ just given $\{G_y(x_1, \ldots, x_m, v)\}_{y\in[n]}$. The security of randomized encoding guarantees that this distribution is simulatable just given the output Y.

Now, one could naively compute n bounded FMS-MPC protocols in parallel to determine the outputs of $G_1, \ldots, G_n$. However, the total number of first round messages would now depend on $|C|$, violating first message succinctness. To circumvent this issue, we delay the computation of those first round messages to the second round. Following the GGM approach, we define a complete binary tree based on the circuit being computed. This tree will have n leaves in total and will be of depth $d = \log(n)$. The y'th leaf is associated with the randomized encoding G_y. Each internal node is associated with an expansion circuit E. This circuit takes as input $(x_1, \ldots, x_m, v)$ and some additional secret randomness, and generates two sets of fresh first round messages, one for each child node. By computing all the expansion circuits using bounded FMS-MPC, we generate a set of fresh first round messages for each leaf node, enabling computation of all randomized encoding circuits using n more bounded FMS-MPC instances. Furthermore, since the CRS of the bounded FMS-MPC has unbounded reusability, it can be used by each node computation in this tree.

To fully compute this tree, each party needs to output its second round message for each node computation in each level, and read all other parties' second round messages. This allows it to recover a new set of first round messages which is required for node computations in the next level. If we implement this protocol naively, the number of rounds in total would match the depth of the tree. Nonetheless, one can still compress it to just two rounds by repeatedly applying the round collapsing transformation: In the first round, each party i outputs its first round message of a bounded FMS-MPC for computing the first expansion circuit (root node). In the second round, party i first outputs its second round message for this bounded FMS-MPC. Then for each level $k \in [2, d-1]$, party i outputs 2^{k-1} garbled circuits which realizes its MPC_2 functionality at this level. That is, for each $y \in [2^{k-1}]$, it computes a garbled circuit of $\mathsf{MPC}_2(\mathsf{E}, (\cdot\,,\, \mathsf{CRS}^*), \cdot\,, \cdot\,)$. This circuit hardwires the description of E and the part CRS^*. It takes as input the part CRS', party i's first round state and all first round messages for computing the y^{th} expansion circuit in this level, and outputs its second round message. In the last level, for each $y \in [n]$, party i computes a garbled circuit of $\mathsf{MPC}_2(\mathsf{F}_y, (\cdot\,,\, \mathsf{CRS}^*), \cdot\,, \cdot\,)$, where F_y computes the randomized encoding G_y. These garbled circuits constitute party i's second round message.

In order to recover the input labels for each garbled circuit, we ask each expansion circuit E to output the input labels which correspond to the correct inputs for each party's next garbled circuit. Each party will actually output encryptions of all input labels along with each garbled circuit, and each expansion circuit will output keys that can be used to decrypt only the correct input labels for each party's next garbled circuit.

It is worth noting that this use of "tree of MPC messages" differs somewhat from how it is used in [AJJM20, BGMM20]. In particular, we build a tree of polynomial size. In order to compute a single large circuit in the second round, each party releases a garbled circuit for each node in the tree. During output reconstruction, the *entire* tree is evaluated. In [AJJM20, BGMM20], to obtain reusability, they set up a implicit tree of *exponential* size. Each time the parties wish to compute a circuit in the second round, they each release a sequence of garbled circuits that trace one root to leaf path in this exponentially-sized tree.

As a final point, since the size of the CRS in a FMS-MPC should only depend on the security parameter λ, we must argue that the CRS we are using is small. Notice that for every node in this tree, either the expansion circuit E or some randomized encoding G_y is computed. The size of either circuit only depends on λ. Therefore it suffices to set $L = \text{poly}(\lambda)$ for some fixed polynomial when instantiating the bounded FMS-MPC. As a result, the CRS only depends on λ, which is what is required for FMS-MPC.

Applying this transformation, we build a FMS-MPC (described in protocol 3) from bounded FMS-MPC.

Protocol 3 (FMS-MPC)**.**
Let $(\mathsf{Gen}, \mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$ *be a bounded FMS-MPC protocol,* $(\mathsf{Garble}, \mathsf{GEval})$ *be a garbling scheme,* $(\mathsf{LabEnc}, \mathsf{LabDec})$ *be a label encryption scheme and* $(\mathsf{CRE.Enc}, \mathsf{CRE.Dec})$ *be a computational randomized encoding scheme. Let* $\mathsf{PRG} = (\mathsf{G}_0, \mathsf{G}_1, \mathsf{H}_0, \mathsf{H}_1)$ *be a length quadrupling PRG. The expansion circuit* E *is defined in algorithm* 5.4 *and circuit* F_y *is defined in algorithm* 5.5. *Let* $L = max(|\mathsf{E}|, |\mathsf{F}_y|)$ *and* $\mathsf{CRS} := (\mathsf{CRS}', \mathsf{CRS}^*) \leftarrow \mathsf{Gen}(1^\lambda, L)$.

- $\mathsf{FMS.MPC}_1\left(1^\lambda, \mathsf{CRS}', i, x_i\right)$:
  1. *Sample* $(r_i, v_i) \leftarrow \{0,1\}^\lambda \times \{0,1\}^\lambda$ *and compute* $\left(\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)}\right) \leftarrow \mathsf{MPC}_1\left(1^\lambda, \mathsf{CRS}', i, ((x_i, v_i), r_i)\right)$.
  2. *Set* $\mathsf{FMS.st}_i^{(1)} := \left(\mathsf{st}_i^{(1)}, r_i, v_i\right)$ *and* $\mathsf{FMS.msg}_i^{(1)} := \mathsf{msg}_i^{(1)}$.
- $\mathsf{FMS.MPC}_2\left(C, \mathsf{CRS}, \mathsf{FMS.st}_i^{(1)}, \left\{\mathsf{FMS.msg}_j^{(1)}\right\}_{j\in[m]}\right)$:
  1. *Compute* $[G_y]_{y\in[n]} \leftarrow \mathsf{CRE.Enc}\left(1^\lambda, C\right)$.
  2. *Define a complete binary tree of depth* $d = \log(n)$ *with* n *leaves. Associate the* y^{th} *leaf with the randomized encoding* G_y.
  3. *Let* $r_i^{(k,y)}$ *denotes party* i*'s secret randomness for computing the* y^{th} *node at level* k. *Set* $r_i^{(1,1)} := r_i$*; compute* $r_i^{(2,1)} := \mathsf{G}_0\left(r_i^{(1,1)}\right)$, $r_i^{(2,2)} := \mathsf{G}_1\left(r_i^{(1,1)}\right)$.
  4. *Compute* $k_i^{(2,1)} := \mathsf{H}_0\left(r_i^{(1,1)}\right)$, $k_i^{(2,2)} := \mathsf{H}_1\left(r_i^{(1,1)}\right)$.
  5. *Compute* $\mathsf{msg}_i^{(2)} \leftarrow \mathsf{MPC}_2\left(\mathsf{E}, \mathsf{CRS}, \mathsf{st}_i^{(1)}, \left\{\mathsf{msg}_j^{(1)}\right\}_{j\in[m]}\right)$.
  6. *For each level* $k \in [2, d-1]$ *and for each* $y \in [1, 2^{k-1}]$*:*
     (a) *Compute* $\left(\widetilde{C}_i^{(k,y)}, \overline{\boldsymbol{lab}}_i^{(k,y)}\right) \leftarrow \mathsf{Garble}\left(1^\lambda,\ \mathsf{MPC}_2\left(\mathsf{E},\ (\cdot\ ,\ \mathsf{CRS}^*),\ \cdot\ ,\ \cdot\ \right)\right)$.
     (b) *Compute* $\overline{\boldsymbol{elab}}_i^{(k,y)} \leftarrow \mathsf{LabEnc}\left(\overline{K}_i^{(k,y)}, \overline{\boldsymbol{lab}}_i^{(k,y)}\right)$*, where* $\overline{K}_i^{(k,y)} = \mathsf{PRF}\left(k_i^{(k,y)}, (t,b)\right)_{t\in[l], b\in\{0,1\}}$.
     (c) *Compute* $r_i^{(k+1,2y-1)} := \mathsf{G}_0\left(r_i^{(k,y)}\right)$, $r_i^{(k+1,2y)} := \mathsf{G}_1\left(r_i^{(k,y)}\right)$*;* $k_i^{(k+1,2y-1)} := \mathsf{H}_0\left(r_i^{(k,y)}\right)$, $k_i^{(k+1,2y)} := \mathsf{H}_1\left(r_i^{(k,y)}\right)$.
  7. *In the last level* d*, for each* $y \in [n]$*:*
     (a) *Compute* $\left(\widetilde{C}_i^{(d,y)}, \overline{\boldsymbol{lab}}_i^{(d,y)}\right) \leftarrow \mathsf{Garble}\left(1^\lambda,\ \mathsf{MPC}_2\left(\mathsf{F}_y,\ (\cdot\ ,\ \mathsf{CRS}^*),\ \cdot\ ,\ \cdot\ \right)\right)$*;*
     (b) *Compute* $\overline{\boldsymbol{elab}}_i^{(d,y)} \leftarrow \mathsf{LabEnc}\left(\overline{K}_i^{(d,y)}, \overline{\boldsymbol{lab}}_i^{(d,y)}\right)$ *where* $\overline{K}_i^{(d,y)} = \mathsf{PRF}\left(k_i^{(d,y)}, (t,b)\right)_{t\in[l], b\in\{0,1\}}$.
  8. *Set* $\mathsf{FMS.msg}_i^{(2)} := \left(\mathsf{msg}_i^{(2)}, \left\{\widetilde{C}_i^{(k,y)}, \overline{\boldsymbol{elab}}_i^{(k,y)}\right\}_{k\in[2,d], y\in[1,2^{k-1}]}\right)$.

- $\mathsf{FMS.MPC}_3\left(\left\{\mathsf{FMS.msg}_j^{(2)}\right\}_{j\in[m]}\right)$:

  1. *Compute* $\left\{\left(\widehat{K}_i^{(2,1)},\ \widehat{K}_i^{(2,2)}\right)\right\}_{i\in[m]} \leftarrow \mathsf{MPC}_3\left(\left\{\mathsf{msg}_j^{(2)}\right\}_{j\in[m]}\right)$.
  2. *For each level* $k \in [2, d]$ *and each* $y \in [1, 2^{k-1}]$*:*
     (a) *For each* $j \in [m]$*:*
        i. *Compute* $\widehat{\boldsymbol{lab}}_j^{(k,y)} \leftarrow \mathsf{LabDec}\left(\widehat{K}_j^{(k,y)}, \overline{\boldsymbol{elab}}_j^{(k,y)}\right)$*;*
        ii. *Compute* $\left(\mathsf{msg}_j^{(2),(k,y)}\right) \leftarrow \mathsf{GEval}\left(\widetilde{C}_j^{(k,y)}, \widehat{\boldsymbol{lab}}_j^{(k,y)}\right)$.
     (b) *If* $k < d$*, then compute*
        $\left\{\left(\widehat{K}_i^{(k+1,2y-1)},\ \widehat{K}_i^{(k+1,2y)}\right)\right\}_{i\in[m]} \leftarrow \mathsf{MPC}_3\left(\left\{\mathsf{msg}_j^{(2),(k,y)}\right\}_{j\in[m]}\right)$.
     (c) *If* $k = d$*, compute*
        $G_y\left((x_1, \dots, x_m), v\right) \leftarrow \mathsf{MPC}_3\left(\left\{\mathsf{msg}_j^{(2),(d,y)}\right\}_{j\in[m]}\right)$.
  3. *Set* $Y \leftarrow \mathsf{CRE.Dec}\left(1^\lambda, C, \{G_y((x_1, \dots, x_m), v)\}_{y\in[n]}\right)$.

Algorithm 5.4 (Circuit E)**.**
Input: $\{(x_j, v_j), r_j\}_{j\in[m]}$.
Hardwired inputs: Description of a length-quadruple $\mathsf{PRG} : (\mathsf{G}_0, \mathsf{G}_1, \mathsf{H}_0, \mathsf{H}_1)$.

1. *For each* $i \in [m]$ *(Generating the left child):*
   (a) *Compute* $\mathsf{CRS'}^{\,0} \leftarrow \mathsf{Gen}\left(1^\lambda\right)$.
   (b) *Compute* $\left(\mathsf{st}_i^{(1),0}, \mathsf{msg}_i^{(1),0}\right) \leftarrow \mathsf{MPC}_1\left(1^\lambda, \mathsf{CRS'}^{\,0}, i, ((x_i, v_i), \mathsf{G}_0(r_i))\right)$.
2. *For each* $i \in [m]$*:*
   (a) *Set* $k_i^0 := \mathsf{H}_0(r_i)$*,* $z_i^0 := \left(\mathsf{CRS'}^{\,0}, \mathsf{st}_i^{(1),0}, \left\{\mathsf{msg}_j^{(1),0}\right\}_{j\in[m]}\right)$.
   (b) *Let* $l := |z_i^0|$*. For* $t \in [l]$*, set* $K_{i,t}^0 := \mathsf{PRF}\left(k_i^0, (t, z_i^0[t])\right)$.
   (c) *Set* $\widehat{K}_i^0 := \left\{K_{i,t}^0\right\}_{t\in[l]}$.
3. *For each* $i \in [m]$ *(Generating the right child):*
   (a) *Compute* $\mathsf{CRS'}^{\,1} \leftarrow \mathsf{Gen}\left(1^\lambda\right)$.
   (b) *Compute* $\left(\mathsf{st}_i^{(1),1}, \mathsf{msg}_i^{(1),1}\right) \leftarrow \mathsf{MPC}_1\left(1^\lambda, \mathsf{CRS'}^{\,1}, i, ((x_i, v_i), \mathsf{G}_1(r_i))\right)$.
4. *For each* $i \in [m]$*:*
   (a) *Set* $k_i^1 := \mathsf{H}_1(r_i)$*,* $z_i^1 := \left(\mathsf{CRS'}^{\,1}, \mathsf{st}_i^{(1),1}, \left\{\mathsf{msg}_j^{(1),1}\right\}_{j\in[m]}\right)$.
   (b) *Let* $l := |z_i^1|$*. For* $t \in [l]$*, set* $K_{i,t}^1 := \mathsf{PRF}\left(k_i^1, (t, z_i^1[t])\right)$.
   (c) *Set* $\widehat{K}_i^1 = \left\{K_{i,t}^1\right\}_{t\in[l]}$.
5. *Output* $\left\{\widehat{K}_i^0\right\}_{i\in[m]}$ *and* $\left\{\widehat{K}_i^1\right\}_{i\in[m]}$.

Algorithm 5.5 (Circuit F_y).
Input: $\{(x_j, v_j), r_j\}_{j\in[m]}$.
Hardwired input: the randomized encoding G_y.

1. *Set* $v := v_1 \oplus \cdots \oplus v_m$.
2. *Output* $G_y((x_1, \ldots, x_m), v)$.

In Appendix G, we prove the following theorem, which, combined with Corollary 5.3, gives the following corollary.

Theorem 5.6. *Assuming a maliciously-secure bounded FMS-MPC protocol, there exists a maliciously-secure FMS-MPC protocol in the CRS model.*

Corollary 5.7. *Assuming LPN with inverse polynomial error rate, there exists a maliciously-secure FMS-MPC protocol in the CRS model.*

5.3 FMS-MPC $\rightarrow$ Reusable two-round MPC

It has been shown in previous work [BGMM20] that any maliciously-secure FMS-MPC protocol in the CRS model implies a maliciously-secure reusable two-round MPC protocol in the CRS model. Thus, we immediately have the following theorem.

Theorem 5.8. *Assuming LPN with inverse polynomial error rate, there exists a maliciously-secure reusable two-round MPC protocol in the CRS model.*

5.3.1 The semi-honest case

We have presented all of our results in this section in the malicious-security setting, which requires a CRS. However, we remark here that we can also achieve semi-honest secure reusable MPC in the plain model from LPN. In fact, we claim that any maliciously-secure reusable MPC in the CRS model plus semi-honest secure vanilla two-round MPC in the plain model implies a semi-honest secure reusable MPC in the plain model. As this transformation is nearly identical to that of [BGMM20, Section 5], we do not provide a formal proof, but give the following sketch.

The vanilla two-round MPC can be used to compute a CRS and first round messages of the reusable MPC, and release *garbled labels* of the CRS and first round messages to all parties. In the second round, each party also releases a garbled circuit that computes their second round message of the reusable MPC. Anyone can combine the labels for the CRS and labels for the first round messages with these garbled circuits to compute the second round messages and thus the output.

References

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In Bernard Chazelle, editor, *ICS 2011*, pages 45–60. Tsinghua University Press, January 2011.

[AIK05] B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynomials and their applications (extended abstract). In *20th Annual IEEE Conference on Computational Complexity (CCC'05)*, pages 260–274, 2005.

[AIK09] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. *Journal of Cryptology*, 22(4):429–469, October 2009.

[AJJM20] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 28–57. Springer, 2020.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 19*, pages 291–308. ACM Press, 2019.

[BCG$^+$19b] 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.

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

[BGMM20] James Bartusek, Sanjam Garg, Daniel Masny, and Pratyay Mukherjee. Reusable two-round mpc from ddh. TCC, 2020.

[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.

[BL20] Fabrice Benhamouda and Huijia Lin. Multiparty reusable non-interactive secure computation. TCC, 2020.

[DGH$^+$20] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. Two-round oblivious transfer from cdh or lpn. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 768–797, Cham, 2020. Springer International Publishing.

[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.

[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.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. 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.

[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.

[GS18a] 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.

[GS18b] 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.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently, 2003.

[JKPT12] Abhishek Jain, Stephan Krenn, Krzysztof Pietrzak, and Aris Tentes. Commitments and efficient zero-knowledge proofs from learning parity with noise. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 663–680. Springer, Heidelberg, December 2012.

[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.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.

A Conforming Protocol

We recall the following specification of conforming protocol from [BGMM20] [GS18b]:
For any m-party MPC protocol Φ between parties $P_1, \ldots, P_m$ with respective inputs $(x_1, \ldots, x_m)$, a conforming protocol is defined by a tuple of functions $(\mathsf{pre}, \mathsf{post})$ and a number of computation steps (actions) $(\phi_1, \ldots, \phi_T)$. The protocol Φ proceeds in three stages: the pre-processing stage, the computation stage and the output stage.

Remark A.1. *We outline some differences between our definition and that of [GS18b]:*

- *Whereas in [GS18b] each party maintains its secret state in the computation phase, in our specification, each party maintains a public state (its computation tape) in the computation phase. Every party initializes its tape in the preprocessing stage, and updates it in each step in the computation stage. In all round $t \in T$, we require that every party's computation tape is the same (hence public).*
- *Without loss of generality, we set party i's secret state (denoted as v_i) to be its input masking bits concatenated with its masking bits for inner gates in its computation. Looking ahead, in the context of multiparty NISC, each party's masking bits v_i can deduced from its secret parameter s_i.*

We begin with some notations and description of parameters. Then we specify each stage in the conforming protocol.

- **Parameters**: Let T be the total number of rounds in Φ and let p be the number of times that some party needs to compute some circuit of fixed size. Let q be the number of steps (gates) to compute any such circuit. Therefore, each party needs to perform pq steps of computation in total, and the total number of rounds is $T = pqm$. Denote by l the length of each x_i, assuming their lengths are all equal. We sometimes appeal to [GS18b] and use the notation v_f to denote $\mathsf{v}_i[f]$ when the index i is clear from the context.
- **Preprocessing stage**: For each $i \in [m]$, party i computes
$$(z_i, \mathsf{v}_i) \leftarrow \mathsf{pre}\left(1^\lambda, i, x_i\right);$$
where the randomized algorithm pre samples $r_i \leftarrow \{0,1\}^l$ and $v_{i,k} \leftarrow \{0,1\}^{q-1}$ for $k \in [p]$. It then sets $z_i := x_i \oplus r_i$ and $\mathsf{v}_i = r_i||v_{i,1}||0|| \ldots ||v_{i,p}||0$. Then party i broadcasts the value z_i and keeps the secret state v_i to itself.
- **Computation stage**: For each $i \in [m]$, party i does the following:
  1. Initialize a computation tape (public state) st_i, which begins with the values $\mathsf{st}_i := z_1||0^{pq}|| \ldots ||z_m||0^{pq}$. For all rounds $t \in [T]$, $\mathsf{st}_1 = \mathsf{st}_2 = \cdots = \mathsf{st}_m$ always.
  2. For each $t \in [T]$, proceed as follows:
     (a) Parse the action ϕ_t as $\phi_t := (i^*, f, g, h)$, where $i^* \in [m]$, and $f, g, h \in [pq + l]$. We sometimes refer to P_{i^*} as the speaking party.
     (b) Read two values α and β from its computation tape: first locate the portion '$z_{i^*}|| \ldots$' on the tape, and then set α to be the value at the f^{th} position starting from z_{i^*}, and β to be the value at the g^{th} position starting from z_{i^*}. In other words, $\alpha := \mathsf{st}_i[(i^* - 1)\,(pq + l) + f]$, $\beta := \mathsf{st}_i[(i^* - 1)\,(pq + l) + g]$.
     (c) If $i = i^*$, then set $\mathsf{v} := \mathsf{v}_{i^*}$ and compute one NAND gate as follows: $\gamma_t = \mathsf{NAND}\left(\mathsf{v}_f \oplus \alpha, \mathsf{v}_g \oplus \beta\right) \oplus \mathsf{v}_h$. Then broadcast the value γ_t to every other party.
     (d) If $i \neq i^*$, then record the value γ_t on its computation tape as follows: first locate the portion '$z_{i^*}|| \ldots$' on the tape, and then write the value γ_t on the h^{th} position starting from z_{i^*}. In other words, set $\mathsf{st}_i[(i^* - 1)\,(pq + l) + h] := \gamma_t$.
- **Output stage**: Let $\mathsf{Z} := (\gamma_1, \ldots, \gamma_T)$ be the transcript of the conforming protocol Φ. Given Z, anyone can decode the output by computing $Y \leftarrow \mathsf{post}(\mathsf{Z})$.

Lemma A.2. *Any MPC protocol can be written as a conforming protocol Φ with the same correctness and security guarantee of the original protocol.*

Proof. Following the proof strategy of [GS18b], without loss of generality consider any MPC protocol where in each round, one party broadcasts one bit that is obtained by computing a circuit on its initial state and the messages it has received so far from other parties. Assume that each party computes p such circuits in total, and each circuit only outputs a single bit. Let each circuit only consists of NAND gates and let q be the number of gates in this

circuit. We show that the above construction correctly executes this MPC protocol:
Correctness: To show correctness, it suffices to argue that all the messages in the original MPC protocol is included in the final transcript Z and they are correctly computed. We use induction to prove this. In the base case, some party i computes its first circuit over its own inputs x_i. The masking bits that it uses are $\mathsf{v} := r_i||v_{i,1}||0$. Consider to cases: firstly, if the action ϕ_t computes an NAND gate over two input bits $(x_i[f], x_i[g])$, then by construction, $\alpha := x_i[f] \oplus \mathsf{v}_f$ and $\beta := x_i[g] \oplus \mathsf{v}_g$, thus $\gamma_t = \mathsf{NAND}\left(x_i[f] \oplus \mathsf{v}_f \oplus \mathsf{v}_f, x_i[g] \oplus \mathsf{v}_g \oplus \mathsf{v}_g\right) \oplus \mathsf{v}_h = \mathsf{NAND}\left(x_i[f], x_i[g]\right) \oplus \mathsf{v}_h$. Therefore it is correctly computed and masked. Secondly, if the action ϕ_t computes an NAND gate over outputs of two inner gates (let the output be y_f and y_g, masked by v_f and v_g previously), then by construction, $\alpha := y_f \oplus \mathsf{v}_f$ and $\beta := y_g \oplus \mathsf{v}_g$, thus $\gamma_t = \mathsf{NAND}\left(y_f \oplus \mathsf{v}_f \oplus \mathsf{v}_f, y_g \oplus \mathsf{v}_g \oplus \mathsf{v}_g\right) \oplus \mathsf{v}_h = \mathsf{NAND}\left(y_f, y_g\right) \oplus \mathsf{v}_h$. Therefore it is correctly computed and masked. Since the last masking bit is 0, the output of this circuit is given in clear and is included in the transcript.
Now for the inductive step, let n be the total number of circuits computed so far and let party i compute the $(n+1)^{th}$ circuit based on the previous messages (which are recorded in st_i by induction hypothesis). We can again use the same argument to conclude that this circuit is also corrected computed.
Finally, the post algorithm reads the output of each circuit from the transcript Z and decodes the output y according to the specification of the original MPC protocol.
Security: To argue security, it suffices to observe that the value of each gate (except the output gate) is masked with a random bit. Therefore, no other information is leaked except for the output of each individual circuit. □

B The GS18 Compiler

The GS18 compiler [GS18b] compresses a multi-round conforming protocol to two rounds. Thus it can be viewed as a publicly decodable two-round MPC protocol and thus we adopt the syntax of two-round MPC to describe this compiler.

Protocol 1 (GS18 Compiler)**.**

- $\left(\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)}\right) \leftarrow \mathsf{MPC}_1\left(1^\lambda, \mathsf{CRS}, C, i, x_i, r_i\right)$:
  1. *Let* Φ *be a conforming protocol for computing* C.
  2. *Party* i *computes* $(z_i, \mathsf{v}_i) \leftarrow \mathsf{pre}\left(1^\lambda, i, x_i\right)$ *(*pre *uses* r_i *as its random coins).*
  3. *For any round* $t \in [T]$ *such that* $i = i^*$ *in the action* $\phi_t = (i^*, f, g, h)$, *set* $\mathsf{v} = \mathsf{v}_i$ *and compute four* OT_1 *messages:* $\{\mathsf{OT}_1\left(1^\lambda, \mathit{NAND}(\mathsf{v}_f \oplus \alpha, \mathsf{v}_g \oplus \beta) \oplus \mathsf{v}_h\right)\}_{\alpha,\beta\in\{0,1\}}$. *Denote each message by* $\mathsf{OT}_1^{(\alpha,\beta),(i,t)}$ *and the secret randomness used to generate it by* $\omega_{\alpha,\beta}^{i,t}$.
  4. *Set* $\mathsf{msg}_i^{(1)} := \left(z_i, \{\mathsf{OT}_1^{(\alpha,\beta),(i,t)}\}_{\{t\}^{i=i^*},\alpha,\beta\in\{0,1\}}\right)$.
  5. *Set* $\mathsf{st}_i^{(1)} := \left(\mathsf{v}_i, \{\omega_{\alpha,\beta}^{i,t}\}_{\{t\}^{i=i^*},\alpha,\beta\in\{0,1\}}\right)$.

- $\mathsf{msg}_i^{(2)} \leftarrow \mathsf{MPC}_2\left(C,\ \mathsf{st}_i^{(1)}, \left\{\mathsf{msg}_j^{(1)}\right\}_{j\in[m]}\right)$:
  1. *Party* i *initializes its computation tape* $\mathsf{st}_i := z_1||0^{pq}||\ldots||z_m||0^{pq}$. *Let* $n := |\mathsf{st}_i|$.
  2. *Set* $\overline{\boldsymbol{lab}}^{i,T+1} := \{\boldsymbol{lab}_{k,0}^{i,T+1}, \boldsymbol{lab}_{k,1}^{i,T+1}\}_{k\in[n]}$ *where for each* $k \in [n]$ *and* $b \in \{0,1\}$ $\boldsymbol{lab}_{k,b}^{i,T+1} = 0^\lambda$.
  3. *For each* t *from* T *to 1, compute* $(\widetilde{\mathsf{P}}^{i,t}, \overline{\boldsymbol{lab}}^{i,t}) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{P}^{i,t}\right)$. *where the circuit* $\mathsf{P}^{i,t}$ *hardcodes all input labels of* $\mathsf{P}^{i,t+1}$ *(see B.1).*
  4. *Set* $\widehat{\boldsymbol{lab}}^{i,1} := \{\boldsymbol{lab}_{k,\mathsf{st}_{i,k}}^{i,1}\}_{k\in[n]}$
  5. *Set* $\mathsf{msg}_i^{(2)} := \left(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \widehat{\boldsymbol{lab}}^{i,1}\right)$.

- $Y \leftarrow \mathsf{MPC}_3\left(\left\{\mathsf{msg}_j^{(2)}\right\}_{j\in[m]}\right)$:
  1. *For each* t *from* 1 *to* T, *do:*
     (a) *Parse action* $\phi_t := (i^*, f, g, h)$.
     (b) *Compute* $\left(\gamma_t, \omega_{\alpha,\beta}^{i^*,t}, \widehat{\boldsymbol{lab}}^{i^*,t+1}\right) := \mathsf{GEval}\left(\widetilde{\mathsf{P}}^{i^*,t}, \widehat{\boldsymbol{lab}}^{i^*,t}\right)$.
     (c) *For each* $j \neq i^*$, *do:*
        i. *Compute* $\left(\mathsf{OT}_2^{(\alpha,\beta),(j,t)}, \widehat{\boldsymbol{lab}}^{j,t+1}\right) = \mathsf{GEval}\left(\widetilde{\mathsf{P}}^{j,t}, \widehat{\boldsymbol{lab}}^{j,t}\right)$.
        ii. *Parse* $\widehat{\boldsymbol{lab}}^{j,t+1} := \left\{\boldsymbol{lab}_{k,\mathsf{st}_{j,k}}^{j,t+1}\right\}_{k\in[n]/\{h\}}$.
        iii. *Recover* $\boldsymbol{lab}_h^{j,t+1} \leftarrow \mathsf{OT}_3\left(\mathsf{OT}_2^{(\alpha,\beta),(j,t)}, \omega_{\alpha,\beta}^{i^*,t}\right)$.
        iv. *Reset* $\widehat{\boldsymbol{lab}}^{j,t+1} := \left\{\{\boldsymbol{lab}_{k,\mathsf{st}_{j,k}}^{j,t+1}\}_{k\in[n]/\{h\}}, \boldsymbol{lab}_h^{j,t+1}\right\}$.
  2. *Let* $\mathsf{Z} := (\gamma_1, \ldots, \gamma_T)$, *set* $Y := \mathsf{post}(\mathsf{Z})$.

Algorithm B.1 (Circuit $\mathsf{P}^{i,t}$)**.**
Input: st_i.
Hardwired inputs: Party i's masking bits (secret state) v_i, its secret randomness for generating OT_1 messages: $\{\omega^{i,t}_{\alpha,\beta}\}_{\{t\}^{i=i^*},(\alpha,\beta)\in\{0,1\}}$. Every other party j's OT_1 messages: $\{\mathsf{OT}_1^{(\alpha,\beta),(j,t)}\}_{\{t\}^{j=i^*},(\alpha,\beta)\in\{0,1\}}$. The input labels of the next garbled circuit $\widetilde{P}^{i,t+1}$: $\overline{\boldsymbol{lab}}^{i,t+1}$. The round action ϕ_t and the description of a two round OT protocol.

1. *Parse $\phi_t := (i^*, f, g, h)$.*
2. *Set $\alpha := \mathsf{st}_i[(i^*-1)(pq+l)+f]$, $\beta := \mathsf{st}_i[(i^*-1)(pq+l)+g]$.*
3. *If $i = i^*$, then:*
   (a) *Set $\mathsf{v} := \mathsf{v}_i$, and compute $\gamma_t = \mathit{NAND}(\mathsf{v}_f \oplus \alpha, \mathsf{v}_g \oplus \beta) \oplus \mathsf{v}_h$.*
   (b) *Set $\mathsf{st}_i[(i^*-1)(pq+l)+h] := \gamma_t$.*
   (c) *Set $\widehat{\boldsymbol{lab}}^{i,t+1} := \left\{\boldsymbol{lab}^{i,t+1}_{k,\mathsf{st}_{i,k}}\right\}_{k\in[n]}$.*
   (d) *Output $\left(\gamma_t, \omega^{i,t}_{\alpha,\beta}, \widehat{\boldsymbol{lab}}^{i,t+1}\right)$.*
4. *If $i \neq i^*$, then:*
   (a) *Compute $\mathsf{OT}_2^{(\alpha,\beta),(i,t)} \leftarrow \mathsf{OT}_2\left(\mathsf{OT}_1^{(\alpha,\beta),(i^*,t)}, \boldsymbol{lab}^{i,t+1}_{h,0}, \boldsymbol{lab}^{i,t+1}_{h,1}\right)$, where the label $\boldsymbol{lab}^{i,t+1}_h$ is the input bit for $\mathsf{st}_i[(i^*-1)(pq+l)+h]$ of its next garbled circuit.*
   (b) *Set $\widehat{\boldsymbol{lab}}^{i,t+1} := \left\{\boldsymbol{lab}^{i,t+1}_{k,\mathsf{st}_{i,k}}\right\}_{k\in[n]/\{h\}}$.*
   (c) *Output $\left(\mathsf{OT}_2^{(\alpha,\beta),(i,t)}, \widehat{\boldsymbol{lab}}^{i,t+1}\right)$.*

C Security Definition of Multiparty Silent NISC

Consider m parties $(P_1, \ldots, P_m)$ with inputs $(x_1, \ldots, x_m)$ respectively that wish to participate in an m-party silent NISC protocol to evaluate any circuit/functionality C over their joint inputs. The description of real/ideal world execution is given below:

Real Execution In the real execution, the m-party silent NISC protocol for computing C is executed in the presence of an adversary $\mathcal{A}$. A trusted dealer generates the CRS, runs the Setup algorithm, and distributes all secret parameters $\{s_i\}_{i\in[m]}$. The adversary $\mathcal{A}$ takes as input the security parameter λ, an auxiliary input z, the CRS and all secret parameters of corrupted parties: $\{s_j\}_{j\in I}$. The honest parties follow the instructions and take as input the CRS and their own secret parameters: $\{s_i\}_{i\in H}$. $\mathcal{A}$ sends the commitments $\{c_j\}_{j\in I}$ and then the messages $\{m_j\}_{j\in I}$ on behalf of the corrupted parties following any arbitrary polynomial-time strategy, and at the end outputs an arbitrary function of its view. We denote $\vec{x} := (x_1, \ldots, x_m)$. The interaction of $\mathcal{A}$ in the protocol defines a random variable $\mathsf{Real}_{\mathcal{A}}(\lambda, \vec{x}, z, I)$ whose value is determined by the coin tosses of the dealer, the adversary, and the honest parties. This random variable contains the output of the adversary along with the output of the honest parties.

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

- **Simulator distributes secret parameters**: The ideal world adversary Sim generates the CRS along with the secret parameters $\{s_i\}_{i\in[m]}$. It then broadcasts the CRS and distributes the secret parameters to each party.
- **Honest parties send inputs to the ideal functionality**: Each honest party sends its input to the ideal functionality. 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 ideal functionality. Let x_i' denote the value sent by P_i. Note that for a semi-honest adversary, $x_i' = x_i$ always.
- **Ideal functionality sends output to the adversary**: The ideal functionality computes $Y \leftarrow C(x_1', \ldots, x_m')$ and sends Y to Sim.

- **Simulator decides whether to continue or abort**: After seeing the value Y, the adversary Sim sends either an abort or continue message to the honest parties. (A semi-honest adversary never aborts.)

The interaction of Sim with the ideal functionality defines a random variable $\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \vec{x}, z, I)$ that contains the outputs of Sim and of the honest parties. Having defined the real and the ideal worlds, we now proceed to define our notion of security.

Definition C.1. *Let λ be the security parameter and let C be any m-party functionality. We say that an m-party silent NISC protocol securely computes C in the presence of malicious/semi-honest adversaries if for every PPT real world malicious/semi-honest adversary $\mathcal{A}$, there exists a PPT ideal world adversary* Sim*, such that for any $\vec{x} \in (\{0,1\}^*)^m$, $z \in \{0,1\}^*$, and $I \subset [m]$, and for any PPT distinguisher D, we have that:*

$$\Pr\left[\ D\left(\mathsf{Real}_{\mathcal{A}}(\lambda, \vec{x}, z, I)\right) = 1\ \right] - \Pr\left[\ D\left(\mathsf{Ideal}_{\mathsf{Sim}}(\lambda, \vec{x}, z, I)\right) = 1\ \right] = \mathrm{negl}(\lambda).$$

D Proof of PCG for GS18 Correlations

Theorem D.1. *Assuming LPN with inverse polynomial noise rate,* $(\mathsf{PCG.Gen}_{\mathsf{GS}}, \mathsf{PCG.Expand}_{\mathsf{GS}})$ *in Protocol* 1 *is a multi-party PCG protocol for GS18 correlations satisfying PCG security in Definition 3.11.*

Proof. We first argue that GS18 correlations are generated properly. That is, considering sender 1 without loss of generality, for any action $\phi_t := (\cdot, f, g, h)$ and $\alpha, \beta \in \{0,1\}$, given $\gamma_{t,(\alpha,\beta)} = \mathsf{NAND}(\mathsf{v}_f \oplus \alpha, \mathsf{v}_g \oplus \beta) \oplus \mathsf{v}_h$ we have that $m^1_{t,(\alpha,\beta),1} = m^1_{t,(\alpha,\beta),0} + \delta_1$ and $m^1_{t,(\alpha,\beta)} = m^1_{t,(\alpha,\beta),\gamma_{t,(\alpha,\beta)}}$.

By our construction of PCG seeds and previous arguments, we know that for degree 1 relation, we have $t^1_{1,0}[i] = t^1_{1,1}[i] + \mathsf{v}_i \cdot \delta_1$ and for degree 2 relation, we have $t^2_{1,0}[i,j] = t^2_{1,1}[i,j] + \mathsf{v}_i\mathsf{v}_j \cdot \delta_1$. For any t, α, β, given $\gamma_{t,(\alpha,\beta)} = \mathsf{NAND}(\mathsf{v}_f \oplus \alpha, \mathsf{v}_g \oplus \beta) \oplus \mathsf{v}_h \equiv (\mathsf{v}_f\mathsf{v}_g) + (\alpha\mathsf{v}_g + \beta\mathsf{v}_f + \mathsf{v}_h) + (\alpha\beta + 1)$, we can express the correlated OT with such choice bit as a sum of three correlated OTs (given below) as happening in our protocol.

- Degree 2 correlated OT: $R^1_0 := \left(\mathsf{v}_f\mathsf{v}_g,\ t^2_{1,0}[f,g]\right)$, $R^1_1 := \left(t^2_{1,1}[f,g],\ t^2_{1,1}[f,g] + \delta_1\right)$.
- Degree 1 correlated OT: $R^2_0 := \left(\alpha\mathsf{v}_g + \beta\mathsf{v}_f + \mathsf{v}_h,\ \alpha \cdot t^1_{1,0}[g] + \beta \cdot t^1_{1,0}[f] + t^1_{1,0}[h]\right)$, $R^2_1 := \left(\alpha \cdot t^1_{1,1}[g] + \beta \cdot t^1_{1,1}[f] + t^1_{1,1}[h],\ \alpha \cdot t^1_{1,1}[g] + \beta \cdot t^1_{1,1}[f] + t^1_{1,1}[h] + \delta_1\right)$.
- Constant relation: $R^3_0 := (\alpha\beta + 1,\ 0)$, $R^3_1 := ((\alpha\beta + 1) \cdot \delta_1,\ \alpha\beta \cdot \delta_1)$.

Next, we argue security. We show that the PCG protocol satisfies the natural extension of Definition 3.11 to the multi-party setting. In particular, given any single party i (either the receiver or one of the senders), their output (expanded) correlation can be reverse sampled given just the output correlations of all other parties.

First consider the case where the adversary corrupts the receiver and senders $j \in [m]/\{i\}$ (all but sender i are corrupted). In this case, we want to show that just given the receiver's expanded correlation R_0, and the set of sender expanded correlations $\{R_j\}_{j \in [m] \setminus \{i\}}$, we can reverse sample the i'th sender's expanded correlation (see Definition 3.11).

The reverse sampling algorithm, given the i'th part $\{m^i_{t,(\alpha,\beta)}\}$ of the receiver's expanded correlations, operates as follows. This defines the ideal distribution.

- Sample a random element $\delta_i \in \mathbb{F}_p$.
- For each $t \in [q]$ and each $\alpha, \beta \in \{0,1\}$, set $m^i_{t,(\alpha,\beta),0} := m^i_{t,(\alpha,\beta)} + \gamma_{t,(\alpha,\beta)} \cdot \delta_i$.
- Output $R_i := \left(\delta_i, \left\{m^i_{t,(\alpha,\beta),0}\right\}_{\alpha,\beta \in \{0,1\}, t \in [q]}\right)$.

Now, by the correctness of generating correlated OTs described above, the real distribution (where sender i's output is computed by expanding its seed) is indistinguishable from the distribution induced by defining sender i's output instead by applying the shift δ_i to messages in the receiver's output. At this point, we can replace the receiver's seeds with simulated ones $\left(\widetilde{k^1_{j,0}}' \leftarrow \mathsf{Sim}(1^\lambda, \mathbb{F}_{2^\lambda}, n'),\ \widetilde{k^2_{j,0}}' \leftarrow \mathsf{Sim}(1^\lambda, \mathbb{F}_{2^\lambda}, n'^2)\right)$, where Sim is the multi-point FSS simulator (see discussion at the end of Section 3.2). Now, the uniformly random shift δ_i is sampled independently of the receiver's seed generation, so we can consider sampling it using the reverse sampling algorithm above (rather than using the δ_i that was sampled before seed generation). Now, we switch the seed generation back to honest, and the resulting distribution is exactly the ideal distribution, where sender i's output is sampled based on the output of the receiver.

Now consider the case where the adversary corrupts all the senders $j \in [m]$. Given the output correlations of each sender, the reverse sampling algorithm for receiver correlations does the following.

- Sample $\mathsf{v} \in \{0,1\}^n$ randomly.
- For each $j \in [m]$, each $t \in [q]$ and each $\alpha, \beta \in \{0,1\}$, compute $\gamma_{t,(\alpha,\beta)}$ from v, and set $m^j_{t,(\alpha,\beta)} := m^j_{t,(\alpha,\beta),0} + \gamma_{t,(\alpha,\beta)} \cdot \delta_j$.
- Output $R_0 = \left(\mathsf{v}, \left\{m^j_{t,(\alpha,\beta)}\right\}_{\alpha,\beta\in\{0,1\},t\in[q],j\in[m]}\right)$.

We show that the ideal distribution induced by the above reverse sampling algorithm is indistinguishable from the real distribution via the following hybrids.

- Hybrid_0: This is the real distribution: namely both $\mathbf{y} \leftarrow \{0,1\}^{n'}$ and $\{\delta_j\}_{j\in[m]} \leftarrow \mathbb{F}^j_{2^\lambda}$ are randomly sampled. Each sender j's seeds are computed as $\left(\cdot, \widetilde{k^1_{j,1}}\right) \leftarrow \mathsf{PCG.Gen}_{\mathsf{sVOLE}}\left(1^\lambda, \widetilde{\mathbf{y}}, \delta_j\right)$; $\left(\cdot, \widetilde{k^2_{j,1}}\right) \leftarrow \mathsf{PCG.Gen}_{\mathsf{sVOLE}}\left(1^\lambda, \widetilde{\mathbf{y}\otimes\mathbf{y}}, \delta_j\right)$. $\mathsf{v} = \mathbf{y} \cdot H_{n',n}$, and all $\left\{m^j_{t,(\alpha,\beta)}\right\}_{\alpha,\beta\in\{0,1\},t\in[q],j\in[m]}$ are computed accordingly.
- $\mathsf{Hybrid}_{j\in[m]}$: Instead of generating sender j's seeds $\left(\widetilde{k^1_{j,1}}, \widetilde{k^2_{j,1}}\right)$ honestly, we replace them one by one with simulated ones: $\left(\widetilde{k^1_{j,1}}' \leftarrow \mathsf{Sim}(1^\lambda, \mathbb{F}_{2^\lambda}, n'),\ \widetilde{k^2_{j,1}}' \leftarrow \mathsf{Sim}(1^\lambda, \mathbb{F}_{2^\lambda}, n'^2)\right)$, where Sim is the multi-point FSS simulator (see discussion at the end of Section 3.2). Indistinguishability follows from the security of multi-point FSS.
- Hybrid_{m+1}: We sample at random the value $\mathsf{v} \leftarrow \{0,1\}^n$ instead of computing it as $\mathsf{v} := \mathbf{y} \cdot H_{n',n}$. Since for all $j \in [m]$, $\left(\widetilde{k^1_{j,1}}', \widetilde{k^2_{j,1}}'\right)$ are independent of $\mathbf{y}$, indistinguishability follows directly from the LPN assumption.

□

E Security of Muliparty Silent NISC

In this section, we prove malicious security of multiparty silent NISC protocol. We begin by listing all the ingredients we are using in this section: let Sim_Φ be a simulator for conforming protocol, Sim_G be the simulator for garbling scheme. Let $\mathcal{A}$ be the real world adversary who corrupts a set of parties $I \subset [m]$. We always assume that this corruption is static.

Description of simulator The ideal world adversary Sim simulates the view of real world adversary $\mathcal{A}$ as follows:

- **Generating secret parameters**: Given a circuit size bound L, the simulator generates the CRS and secret parameter on behalf of an honest dealer: $\mathsf{CRS} \leftarrow \mathsf{Gen}\left(1^\lambda\right)$; $\{s_i\}_{i\in[m]} \leftarrow \mathsf{Setup}\left(1^\lambda, L\right)$. It then gives the CRS and $\{s_j\}_{j\in I}$ to the corrupted parties (controlled by $\mathcal{A}$), and additionally computes $\{\mathsf{v}_j\}_{j\in I}$ accordingly, keeping this value to itself.
- **Simulating commit phase**: The simulator runs $\mathsf{Sim}_\Phi\left(1^\lambda\right)$ and obtains $\{z_i\}_{i\in H}$: (Recall that by our construction $c_i = z_i$). Then it saves the state of Sim_Φ and sends $\{c_i\}_{i\in H}$ on behalf of honest parties to the real world adversary $\mathcal{A}$, who responds with $\{c_j\}_{j\in I}$ on behalf of corrupted parties.
- **Extracting corrupted party inputs**: For each $j \in I$, the simulator parses $\mathsf{v}_j := \mathsf{v}'_j||\mathsf{v}^*_j$. (recall that $\mathsf{v}'_j \in \{0,1\}^l$ is the random mask for party i's input) Then set $x_j := c_j \oplus \mathsf{v}'_j$.
- **Receiving output from ideal functionality**: Each honest party $i \in H$ sends its input x_i to the ideal functionality, and the simulator sends corrupted party inputs $\{x_j\}_{j\in I}$ on behalf of the adversary $\mathcal{A}$. The ideal functionality computes $Y \leftarrow C(x_1, \ldots, x_n)$ and sends Y to the simulator.
- **Simulating conforming protocol**: The simulator continues to run Sim_Φ on inputs $\left(\{x_j\}_{j\in I}, \{\mathsf{v}_j\}_{j\in I}, Y\right)$, and obtain the following:
  - The public computation tape (which is same for all honest parties) $\{\mathsf{st}_i\}_{i\in H}$.
  - The transcript of the computation phase, still denoted by Z. Let Z_t be the bit sent in the t^{th} round of the computation phase.
- **Setting up GS18 correlations for honest parties**: For $t \in [T]$, parse $\phi_t = (i^*, f, g, h)$ and set $\alpha^* := \mathsf{st}_i[(i^*-1)(pq+l)+f]$, $\beta^* := \mathsf{st}_i[(i^*-1)(pq+l)+g]$:
  - If $(i^* \in H)$, then:

1. For any $j \in H/\{i^*\}$, sample two random strings $\left(\widetilde{m}_{t,0}^{i^*,j}, \widetilde{m}_{t,1}^{i^*,j}\right) \leftarrow \{0,1\}^\lambda \times \{0,1\}^\lambda$. For each $\alpha, \beta \in \{0,1\}$, set party i^*'s receiver strings for round t as $\left(\gamma_{t,(\alpha,\beta)}^{i^*} := \mathsf{Z}_t, \widetilde{m}_{t,(\alpha,\beta)}^{i^*,j} := \widetilde{m}_{t,\mathsf{Z}_t}^{i^*,j}\right)$; set party j's sender strings for round t as $\left(\widetilde{m}_{t,(\alpha,\beta),0}^{i^*,j} := \widetilde{m}_{t,0}^{i^*,j}, \widetilde{m}_{t,(\alpha,\beta),1}^{i^*,j} := \widetilde{m}_{t,1}^{i^*,j}\right)$.
2. For any $j \in I$, use s_j to compute party j's sender strings: $\left(\widetilde{m}_{t,(\alpha^*,\beta^*),0}^{i^*,j},\ \widetilde{m}_{t,(\alpha^*,\beta^*),1}^{i^*,j}\right)$. Then for each $\alpha, \beta \in \{0,1\}$ set party i^*'s receiver strings for round t as $\left(\gamma_{t,(\alpha,\beta)}^{i^*} := \mathsf{Z}_t,\ \widetilde{m}_{t,(\alpha,\beta)}^{i^*,j} := \widetilde{m}_{t,(\alpha^*,\beta^*),\mathsf{Z}_t}^{i^*,j}\right)$.

– If $(i^* \in I)$, then:

1. For any $j \in H$, use s_{i^*} to compute party i^*'s receiver strings: $\left(\gamma_{t,(\alpha^*,\beta^*)}^{i^*},\ \widetilde{m}_{t,(\alpha^*,\beta^*)}^{i^*,j}\right)$. Then sample $\widetilde{m}_{t,(\alpha^*,\beta^*),1-\mathsf{Z}_t}^{i^*,j} \leftarrow \{0,1\}^\lambda$, and for each $\alpha, \beta \in \{0,1\}$, set party j's sender strings for round t as: $\left(\widetilde{m}_{t,(\alpha,\beta),\mathsf{Z}_t}^{i^*,j} := \widetilde{m}_{t,(\alpha^*,\beta^*)}^{i^*,j},\ \widetilde{m}_{t,(\alpha,\beta),1-\mathsf{Z}_t}^{i^*,j} := \widetilde{m}_{t,(\alpha^*,\beta^*),1-\mathsf{Z}_t}^{i^*,j}\right)$.

- **Simulating compute phase**: For each $i \in H$, Sim generates message m_i on behalf of party i as follows:
  1. Set $\overline{\mathbf{lab}}^{i,T+1} := \left(\mathbf{lab}_{k,0}^{i,T+1}, \mathbf{lab}_{k,1}^{i,T+1}\right)_{k\in[N]}$ where for each $k \in [N]$ and $b \in \{0,1\}$ $\mathbf{lab}_{k,b}^{i,T+1} = 0^\lambda$.
  2. For each t from T to 1:
     (a) Parse $\phi_t := (i^*, f, g, h)$.
     (b) Set $\alpha^* := \mathsf{st}_i[(i^*-1)(pq+l)+f]$, $\beta^* := \mathsf{st}_i[(i^*-1)(pq+l)+g]$.
     (c) If $i = i^*$, then compute:
     $$\left(\widetilde{\mathsf{P}}^{i,t}, \widehat{\mathbf{lab}}^{i,t}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^{|\mathsf{P}^{i,t}|}, 1^{|\mathsf{st}_i|}, \left(\mathsf{Z}_t, \left\{\widetilde{m}_{t,(\alpha^*,\beta^*)}^{i,j}\right\}_{j\in[m]/\{i\}}, \widehat{\mathbf{lab}}^{i,t+1}\right)\right)$$
     (d) If $i \neq i^*$, then set $\widehat{\mathbf{lab}}^{i,t+1} := \left\{\mathbf{lab}_{k,\mathsf{st}_{i,k}}^{i,t+1}\right\}_{k\in[N]/\{h\}}$, and compute:
     $$\left(\widetilde{\mathsf{P}}^{i,t}, \widehat{\mathbf{lab}}^{i,t}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^{|\mathsf{P}^{i,t}|}, 1^{|\mathsf{st}_i|}, \left(\mathbf{lab}_h^{i,t+1} \oplus \widetilde{m}_{t,(\alpha^*,\beta^*),0}^{i^*,i},\ \mathbf{lab}_h^{i,t+1} \oplus \widetilde{m}_{t,(\alpha^*,\beta^*),1}^{i^*,i},\ \widehat{\mathbf{lab}}^{i,t+1}\right)\right)$$
  3. Set $m_i := \left(\{\mathsf{P}^{i,t}\}_{t\in[T]},\ \widehat{\mathbf{lab}}^{i,1}\right)$.

Now, we prove indistinguishability via the following hybrids.

Hybrid_0 : This hybrid is the real world execution.

Hybrid_1 : This hybrid is the same as Hybrid_0, except that we remove pairwise setup between party $i \in H$ and any other $j \in [m]$ whenever party i is the sender.

For every $j \in [m]$, first sample a random shift $\delta_{j,i} \leftarrow \mathbb{F}_p$. For every $t \in T$ such that $\phi_t := (i^*, f, g, h)$ and $i^* \neq i$, for each $\alpha, \beta \in \{0,1\}$, let $\left(\gamma_t := \gamma_{t,(\alpha,\beta)}^{i^*},\ m_{t,(\alpha,\beta)}^{i^*,i}\right)$ be party i^*'s receiver strings for round t. Note that this is honestly computed from $\mathbf{s}_0^{i^*,i}$. Now set party i's sender strings for round t as $\widetilde{m}_{t,(\alpha,\beta),\gamma_t}^{i^*,i} := \rho\left(m_{t,(\alpha,\beta)}^{i^*,i}\right),\ \widetilde{m}_{t,(\alpha,\beta),1-\gamma_t}^{i^*,i} := \rho\left(m_{t,(\alpha,\beta)}^{i^*,i} + \delta_{i^*,i}\right)$.

Now set $s_i := \left(\left\{\mathbf{s}_0^{i,j}\right\}_{j\in[m]/\{i\}}\right)$ (since we remove pairwise setup between party i and any $j \in [m]$ whenever party i is the sender). Party i hardwires above sender strings in each of its garbled circuit.

This hybrid is indistinguishable from Hybrid_0 due to the security of PCG (Theorem D.1). In particular, indistinguishability follows from the fact that we are reverse sampling each sender string conditioning on the receiver string.

Hybrid_2 : In this hybrid we switch honest party sender strings into truly random strings:

For each $i \in H$ and for each $t \in T$, parse $\phi_t := (i^*, f, g, h)$. If $i^* \neq i$, then we reset party i's sender string for round t as: $\widetilde{m}_{t,(\alpha,\beta),\gamma_t}^{i^*,i} := \widetilde{m}_{t,(\alpha,\beta,\gamma_t)}^{i^*,i}$, and $\widetilde{m}_{t,(\alpha,\beta),1-\gamma_t}^{i^*,i} \leftarrow \{0,1\}^\lambda$ being a truly random string.

This hybrid is indistinguishable from Hybrid_1 due to the security of random oracle (as a correlation robust hash function) In particular, the probability that the adversary will be able to correctly guess $\delta_{i^*,i}$ and make queries $m_{t,(\alpha,\beta)}^{i^*,i} + \delta_{i^*,i}$ to ρ is negligible.

$\mathsf{Hybrid}_{k\in[3,T+3]}$: In this sequence of hybrids, we simulate honest party garbled circuits as follows:

We start by faithfully executing the conforming protocol on honest party inputs and the masking bits and those of the corrupted parties. This yields a transcript of the computation phase, denoted as Z. Let st_i denote the public computation tape for party i. Each Hybrid_k differs from the Hybrid_{k-1} in terms of how each honest party's garbled circuit are generated in round $t := k-3$. We use a sequence of inner hybrids: $\left(\mathsf{Hybrid}_{k,0} := \mathsf{Hybrid}_{k-1},\ \mathsf{Hybrid}_{k,1},\ \mathsf{Hybrid}_{k,2} := \mathsf{Hybrid}_{k}\right)$:

Inner hybrid $\mathsf{Hybrid}_{k,1}$

Parse $\phi_t := (i^*, f, g, h)$, and set $\alpha^* := \mathsf{st}_i[(i^*-1)(pq+l)+f]$, $\beta^* := \mathsf{st}_i[(i^*-1)(pq+l)+g]$,

- If $i^* \in H$, then make the following changes: simulate party i^*'s garbled circuit for round t as:

$$\left(\widetilde{\mathsf{P}}^{i^*,t}, \widehat{\mathbf{lab}}^{i^*,t}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^{|\mathsf{P}^{i^*,t}|}, 1^{|\mathsf{st}_{i^*}|}, \left(\mathsf{Z}_t, \left\{\widetilde{m}^{i^*,j}_{t,(\alpha^*,\beta^*)}\right\}_{j\in[m]/\{i\}}, \widehat{\mathbf{lab}}^{i^*,t+1}\right)\right)$$

  Where $\widehat{\mathbf{lab}}^{i^*,t+1}$ are the honestly generated labels of its next round garbled circuit.

- Otherwise, for all $i \in H$, simulate party i's garbled circuit for round t as:

$$\left(\widetilde{\mathsf{P}}^{i,t}, \widehat{\mathbf{lab}}^{i,t}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^{|\mathsf{P}^{i,t}|}, 1^{|\mathsf{st}_i|}, \left(\mathbf{lab}^{i,t+1}_{h,0} \oplus \widetilde{m}^{i^*,i}_{t,(\alpha^*,\beta^*),0},\ \mathbf{lab}^{i,t+1}_{h,1} \oplus \widetilde{m}^{i^*,i}_{t,(\alpha^*,\beta^*),1},\ \widehat{\mathbf{lab}}^{i,t+1}\right)\right)$$

  Where $\widehat{\mathbf{lab}}^{i,t+1} := \left\{\mathbf{lab}^{i,t+1}_{k,\mathsf{st}_{i,k}}\right\}_{k\in[N]/\{h\}}$ are the honestly generated labels of its next round garbled circuit.

The inner hybrid $\mathsf{Hybrid}_{k,1}$ is indistinguishable from $\mathsf{Hybrid}_{k,0}$ due to the security of garbled circuits.

Inner hybrid $\mathsf{Hybrid}_{k,2}$

- For all $i \in H$, we make the following changes: whenever $i^* \in I$, simulate party i's garbled circuit for round t as:

$$\left(\widetilde{\mathsf{P}}^{i,t}, \widehat{\mathbf{lab}}^{i,t}\right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^{|\mathsf{P}^{i,t}|}, 1^{|\mathsf{st}_i|}, \left(\mathbf{lab}^{i,t+1}_{h} \oplus \widetilde{m}^{i^*,i}_{t,(\alpha^*,\beta^*),0},\ \mathbf{lab}^{i,t+1}_{h} \oplus \widetilde{m}^{i^*,i}_{t,(\alpha^*,\beta^*),1},\ \widehat{\mathbf{lab}}^{i,t+1}\right)\right)$$

This inner hybrid is indistinguishable from $\mathsf{Hybrid}_{k,1}$ since $\widetilde{m}^{i^*,i}_{t,(\alpha^*,\beta^*),1-\mathsf{Z}_t}$ is a uniformly random string.

Hybrid_{T+4} : This hybrid is the same as Hybrid_{T+3} except that we remove pairwise setup between party $i \in H$ and any other $j \in I$ whenever i is the receiver.

For every $i \in H$, directly sample random masking bits $\mathsf{v}_i \leftarrow \{0,1\}^n$. Then for every $t \in T$ such that $\phi_t := (i^*, f, g, h)$ and $i^* = i$, for each $\alpha, \beta \in \{0,1\}$ and for every $j \in [m]$, let $\left(\widetilde{m}^{i,j}_{t,(\alpha,\beta),0},\ \widetilde{m}^{i,j}_{t,(\alpha,\beta),1}\right)$ be party j's sender strings for round t. Note that if $j \in I$, this is honestly computed from $\mathbf{s}^{i,j}_1$, and for all $j \in H/\{i\}$ its sender strings are defined in Hybrid_1. Now set $\mathsf{v} := \mathsf{v}_i$ and honestly compute $\gamma^i_{t,(\alpha,\beta)} := \gamma_t = \mathsf{NAND}(\mathsf{v}_f \oplus \alpha, \mathsf{v}_g \oplus \beta) \oplus \mathsf{v}_h$. Then set party i's receiver strings for round t as $\left(\gamma^i_{t,(\alpha,\beta)},\ \widetilde{m}^{i,j}_{t,(\alpha,\beta)} := \widetilde{m}^{i,j}_{t,(\alpha,\beta),\gamma_t}\right)$.

Now set $s_i := \perp$ (since we remove pairwise setup between party i and any other $j \in [m]$ whenever i is the receiver). Party i hardwires above receiver strings in each of its garbled circuit.

This hybrid is indistinguishable from Hybrid_{T+3} due to the security of PCG (Theorem D.1). In particular, indistinguishability follows from the fact that we are reverse sampling the receiver string conditioning on the sender string.

Hybrid_{T+5} : This hybrid is the same as Hybrid_{T+4} except that we change honest party's all receiver and sender strings as follows:

For any $i \in H$ and each $t \in [T]$, $\alpha, \beta \in \{0,1\}$, parse $\phi_t := (i^*, f, g, h)$. If $i^* = i$, then for each $j \in [m]/\{i\}$, set $\left(\gamma^i_{t,(\alpha,\beta)} := \mathsf{Z}_t,\ \widetilde{m}^{i,j}_{t,(\alpha,\beta)} := \widetilde{m}^{i,j}_{t,(\alpha^*,\beta^*)}\right)$. Otherwise, set $\left(\widetilde{m}^{i^*,i}_{t,(\alpha,\beta),0} := \widetilde{m}^{i^*,i}_{t,(\alpha^*,\beta^*),0},\ \widetilde{m}^{i^*,i}_{t,(\alpha,\beta),1} := \widetilde{m}^{i^*,i}_{t,(\alpha^*,\beta^*),1}\right)$.

This hybrid is indistinguishable from Hybrid_{T+4} since this only incurs a syntactic change.

Hybrid_{T+6} : In this last hybrid we change how the transcript Z_Φ, the public state st_i, and the masking bits $\{\mathsf{v}_i\}_{i\in H}$ are generated. We simulate them by evoking Sim_Φ on input $\{x_j\}_{j\in I}$, $\{\mathsf{v}_j\}_{j\in I}$, as well as the output Y obtained from the ideal functionality.

This is indistinguishable from the previous hybrid due to the security of the conforming protocol. Finally, notice that this hybrid is the same as the ideal world execution.

F Construction of Bounded FMS-MPC

Definition F.1 (Bounded FMS-MPC)**.** *Let* Gen *be an algorithm that generates a* CRS*. We say that the protocol* $\pi^* = (\mathsf{Gen}, \mathsf{BFMS.MPC}_1, \mathsf{BFMS.MPC}_2, \mathsf{BFMS.MPC}_3)$ *is a bounded FMS-MPC protocol if it is a two-round MPC protocol with the following properties:*

- *Bounded circuit size: The* Gen *algorithm takes as input the security parameter* λ*, a circuit size bound* L*, and outputs a* $\mathsf{CRS} := (\mathsf{CRS}', \mathsf{CRS}^*)$*. The size of* CRS' *only depends on an upper bound on input size, whereas the size of* CRS^* *can be as large as* L*. Moreover, the protocol* π^* *only supports circuits such that* $|C| \leq L$*. When the value* CRS^* *is fixed, the* Gen *algorithm only takes as input* λ *and outputs* CRS'*.*
- *Reusable CRS: The part* CRS^* *only needs to be set up once, and can be reused across an unbounded polynomial number of two-round MPC protocols.*
- *First message succinctness: The* $\mathsf{BFMS.MPC}_1$ *algorithm only takes as input* $\left(1^\lambda, \mathsf{CRS}', i, x_i\right)$*. In particular, its runtime should not depend on the circuit size* $|C|$*.*

Correctness and Security The correctness of bounded FMS-MPC is identical to the standard two-round MPC. For security, we slightly modify the real/ideal world paradigm in the following way:

- **Real execution**: A trusted dealer runs Gen on input L (chosen by the adversary) and computes a $\mathsf{CRS} := (\mathsf{CRS}', \mathsf{CRS}^*)$. It then gives CRS to the adversary $\mathcal{A}$. The adversary then specifies an arbitrary polynomial $q(\cdot)$ in security parameter λ of its choice. Let $\pi = (\mathsf{CRS}^*, \mathsf{Gen}, \mathsf{BFMS.MPC}_1, \mathsf{BFMS.MPC}_2, \mathsf{BFMS.MPC}_3)$ be the two-round MPC protocol with the fixed value of CRS^*. For any arbitrary sequence of circuits $\left(C_1, \ldots, C_{q(\lambda)}\right)$ such that each $|C_y|_{y \in [q(\lambda)]} \leq L$, the honest parties follow the instructions of π to compute C_y. $\mathcal{A}$ sends all messages of the protocol on behalf of the corrupted parties following any arbitrary polynomial-time strategy. Let $\vec{x}_y := (x_{y,1}, \ldots, x_{y,m})$ be the inputs to circuit C_y. The interaction of $\mathcal{A}$ in the protocol defines a random variable $\mathsf{Real}_{\mathcal{A}}\left(\lambda, \{\vec{x}_y\}_{y \in q(\lambda)}, z, I\right)$ whose value is determined by the coin tosses of the adversary and the honest parties. This random variable contains the CRS^*, the output of the adversary as well as the outputs of the honest parties for computing each circuit.
- **Ideal execution**: In the ideal execution, the ideal adversary Sim interacts with an ideal functionality as follows:
  - The simulator runs Gen to generate the CRS.
  - For each $y \in q(\lambda)$, the honest parties send its input for circuit C_y to the ideal functionality. Each corrupted party may send arbitrary inputs to the ideal functionality. Semi-honest adversary does not cheat on their inputs.
  - For each $y \in q(\lambda)$, the ideal functionality computes $Y_y \leftarrow C_y\left(\vec{x}_y\right)$ and gives it to Sim.
  - For each $y \in q(\lambda)$, the simulator Sim outputs the view of corrupted parties: $\left\{\left(\mathsf{BFMS.msg}_i^{(1),y}, \mathsf{BFMS.msg}_i^{(2),y}\right)\right\}_{i \in H}$ and the value Y_y obtained from the ideal functionality.

  The interaction of Sim and ideal functionality defines a random variable $\mathsf{Ideal}_{\mathsf{Sim}}\left(\lambda, \{\vec{x}_y\}_{y \in q(\lambda)}, z, I\right)$.

As before, we require that the two distributions of $\mathsf{Real}_{\mathcal{A}}\left(\lambda, \{\vec{x}_y\}_{y \in q(\lambda)}, z, I\right)$ and $\mathsf{Ideal}_{\mathsf{Sim}}\left(\lambda, \{\vec{x}_y\}_{y \in q(\lambda)}, z, I\right)$ are computationally indistinguishable.

Transformation overview It is straightforward to see that a multiparty silent NISC with large reusable CRS implies the following three-round FMS-MPC protocol:

1. The Gen algorithm first sets up the CRS for multiparty silent NISC: $\mathsf{CRS} := (\mathsf{CRS}', \mathsf{CRS}^*)$. Then it sets up the CRS for a maliciously secure vanilla two-round MPC protocol (to be used in the next step) and includes this CRS inside CRS'.
2. Define a circuit F, which takes as input the security parameter λ, the circuit size bound L, every party's input $\{x_i\}_{i \in [m]}$, and the CRS'. It first plays the role of an honest dealer and computes $\{s_i\}_{i \in [m]} \leftarrow \mathsf{Setup}\left(1^\lambda, L\right)$. Then it computes commitments on behalf of each party: $c_i \leftarrow \mathsf{Commit}\left(i, x_i, s_i, \mathsf{CRS}'\right)$. Finally it distributes $\{s_i\}_{i \in [m]}$ to every party and outputs $\{c_i\}_{i \in [m]}$. It is easy to see that this circuit implements the setup and commit phase of a multiparty silent NISC. Correspondingly, in the first and second round of the three-round MPC, we simply use a vanilla two-round MPC protocol to compute the circuit F.

3. By the end of the second round, each party i can recover the output of the vanilla MPC, which contains its secret parameter s_i and all the commitments $\{c_i\}_{i\in[m]}$. Then party i can compute $m_i \leftarrow \mathsf{Compute}\left(i, x_i, s_i, \mathsf{CRS}, \{c_j\}_{j\in[m]}, C\right)$, and broadcasts m_i as its third-round message.
4. By the end of this three-round MPC protocol, anyone can recover the output by computing $Y \leftarrow \mathsf{Recover}\left(\{m_j\}_{j\in[m]}\right)$.

This three-round MPC protocol inherits the circuit size bound and reusable large CRS from the underlying multiparty silent NISC. The first message succinctness comes from the fact that the circuit F only takes as input λ, L, CRS' and $\{x_i\}_{i\in[m]}$. Although the runtime of MPC_1 in the vanilla two-round MPC depends on the size of F, this circuit does not depend on C, and therefore MPC_1 has no dependence on $|C|$. Now in order to obtain a bounded FMS-MPC, we only have to remove one extra round.

To reduce round complexity, we use the round-collapsing approach from [BGMM20]: In the second round, each party i also outputs a garbled circuit that realizes its third-round functionality. That is, we garble the circuit $\mathsf{Compute}\,(i, x_i, \cdot, \mathsf{CRS}, \cdot, C)$, which hardwires the values (x_i, CRS, C) and receives $\left(s_i,\ \{c_j\}_{j\in[m]}\right)$ as inputs. To enable computing each party's garbled circuit, we also modify the circuit F so that it outputs the input labels corresponding to the values $\left(s_i,\ \{c_j\}_{j\in[m]}\right)$ for each party i's garbled circuit. In the real protocol, party i will actually output its garbled circuit along with the encryption of all input labels using a label encryption scheme (see section 3.2): These input labels are encrypted using a $(2 \times n)$ grid of keys, and each key is derived from a PRF. Each party will sample its randomness r_i as its PRF key, and also supply r_i as its input to the circuit F. This enables F to output the keys that can be used to decrypt only the correct input labels for each party's garbled circuit.

Applying this transformation, we build a bounded FMS-MPC (described in protocol 4) from multiparty silent NISC with reusable large CRS.

Protocol 4 (Bounded FMS-MPC).
Let $(\mathsf{Gen}_{\mathsf{NISC}}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Compute}, \mathsf{Recover})$ *be a multiparty silent NISC protocol,* $(\mathsf{Garble}, \mathsf{GEval})$ *be a garbling scheme. Let* $(\mathsf{Gen}_{\mathsf{MPC}}, \mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$ *be a maliciously secure vanilla two-round MPC protocol and* $(\mathsf{LabEnc}, \mathsf{LabDec})$ *be a label encryption scheme. The circuit* F *is defined in algorithm F.2.*

- $\mathsf{Gen}(1^\lambda, L)$*: If* CRS^* *is fixed then compute* $\mathsf{CRS}' \leftarrow \mathsf{Gen}_{\mathsf{NISC}}(1^\lambda)$*, otherwise compute* $(\mathsf{CRS}', \mathsf{CRS}^*) \leftarrow \mathsf{Gen}_{\mathsf{NISC}}\left(1^\lambda, L\right)$*. Also compute* $\widetilde{\mathsf{CRS}} \leftarrow \mathsf{Gen}_{\mathsf{MPC}}\left(1^\lambda\right)$*, and set* $\mathsf{CRS}' := \left(\mathsf{CRS}', \widetilde{\mathsf{CRS}}\right)$ *and* $\mathsf{CRS} := (\mathsf{CRS}', \mathsf{CRS}^*)$.
- $\mathsf{BFMS.MPC}_1\left(1^\lambda, \mathsf{CRS}', i, x_i\right)$:
  1. *Sample* $r_i \leftarrow \{0,1\}^\lambda$*, and compute* $\left(\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)}\right) \leftarrow \mathsf{MPC}_1\left(1^\lambda, \widetilde{\mathsf{CRS}}, \mathsf{F}, i, (x_i, r_i)\right)$.
  2. *Set* $\mathsf{BFMS.st}_i^{(1)} := \left(\mathsf{st}_i^{(1)}, r_i\right)$ *and* $\mathsf{BFMS.msg}_i^{(1)} := \mathsf{msg}_i^{(1)}$.
- $\mathsf{BFMS.MPC}_2\left(C, \mathsf{CRS}, \mathsf{BFMS.st}_i^{(1)}, \left\{\mathsf{BFMS.msg}_j^{(1)}\right\}_{j\in[m]}\right)$:
  1. *Compute* $\mathsf{msg}_i^{(2)} \leftarrow \mathsf{MPC}_2\left(\mathsf{F}, \mathsf{st}_i^{(1)}, \left\{\mathsf{msg}_j^{(1)}\right\}_{j\in[m]}\right)$.
  2. *Compute* $(\widetilde{C}_i, \overline{\boldsymbol{lab}}_i) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{Compute}(i, x_i, \cdot, \mathsf{CRS}, \cdot, C)\right)$.
  3. *Compute* $\overline{\boldsymbol{elab}}_i \leftarrow \mathsf{LabEnc}\left(\overline{K}_i, \overline{\boldsymbol{lab}}_i\right)$ *where* $\overline{K}_i = \mathsf{PRF}\left(r_i, (t,b)\right)_{t\in[n], b\in\{0,1\}}$.
  4. *Set* $\mathsf{BFMS.msg}_i^{(2)} := \left(\mathsf{msg}_i^{(2)}, \widetilde{C}_i, \overline{\boldsymbol{elab}}_i\right)$.
- $\mathsf{BFMS.MPC}_3\left(\left\{\mathsf{BFMS.msg}_j^{(2)}\right\}_{j\in[m]}\right)$:
  1. *Compute* $\left\{\widehat{K}_j\right\}_{j\in[m]} \leftarrow \mathsf{MPC}_3\left(\left\{\mathsf{msg}_j^{(2)}\right\}_{j\in[m]}\right)$.
  2. *For each* $j \in [m]$*:*
     - *Compute* $\widehat{\boldsymbol{lab}}_j \leftarrow \mathsf{LabDec}\left(\widehat{K}_j, \overline{\boldsymbol{elab}}_j\right)$*;*
     - *Compute* $m_j \leftarrow \mathsf{GEval}\left(\widetilde{C}_j, \widehat{\boldsymbol{lab}}_j\right)$.
  3. *Set* $Y \leftarrow \mathsf{Recover}\left(\{m_j\}_{j\in[m]}\right)$.

Algorithm F.2 (Circuit F).
Input: $\{(x_j, r_j)\}_{j\in[m]}$.
Hardwired inputs: the security parameter λ*, a circuit size bound* L*,* CRS' *and description of a* PRF.

1. $\{s_i\}_{i\in[m]} \leftarrow \mathsf{Setup}(1^\lambda, L)$.
2. *For each* $i \in [m]$*, compute* $c_i \leftarrow \mathsf{Commit}(i, x_i, s_i, \mathsf{CRS}')$.
3. *For each* $i \in [m]$*:*
   - *Set* $z_i := \left(s_i, \{c_j\}_{j\in[m]}\right)$ *and let* $n = |z_i|$*, for each* $t \in [n]$*, set* $K_{i,t} = \mathsf{PRF}\left(r_i, (t, z_i[t])\right)$*;*
   - *Set* $\widehat{K}_i = \{K_{i,t}\}_{t\in[n]}$.
4. *Output* $\left\{\widehat{K}_i\right\}_{i\in[m]}$.

Algorithm F.3 (Circuit F')**.**
Input: $\left(\{s_j\}_{j\in I}, \{c_j\}_{j\in[m]}, \{r_j\}_{j\in I}\right)$;
Hardwired input: The description of a PRF.

1. *For each* $j \in I$:
   - *Set* $z_j := \left(s_j, \{c_i\}_{i\in[m]}\right)$;
   - *Set* $n = |z_j|$, *for each* $t \in [n]$, *set* $K_{j,t} = \mathsf{PRF}(r_j, (t, z_j[t]))$;
   - *Set* $\widehat{K}_j = \{K_{j,t}\}_{t\in[n]}$.
2. *For each* $i \in H$:
   - *For each* $t \in [n]$, *randomly sample* $K_{i,t} \leftarrow \{0,1\}^\lambda$;
   - *Set* $\widehat{K}_i = \{K_{i,t}\}_{t\in[n]}$.
3. *Output* $\left\{\widehat{K}_i\right\}_{i\in[m]}$.

I

Security We prove malicious security of this bounded FMS-MPC protocol, relying on the following ingredients: Let $\mathsf{Sim}_{\mathsf{NISC}}$ be a simulator for a multiparty silent NISC protocol, $\mathsf{Sim}_{\mathsf{G}}$ be a simulator for garbling scheme and $\mathsf{Sim}_{\mathsf{MPC}}$ be a simulator for maliciously secure vanilla two round MPC protocol. Let the adversary $\mathcal{A}$ corrupt a set of parties $I \subset [m]$ in a static manner.

Description Of Simulator The ideal world adversary Sim simulates the view of real world adversary as follows:

1. **Sampling CRS**: The adversary chooses a circuit size bound L. Then the simulator samples $(\mathsf{CRS}', \mathsf{CRS}^*) \leftarrow \mathsf{Gen}\left(1^\lambda, L\right)$ and gives it to $\mathcal{A}$.
2. **Adversary decides number of executions**: The adversary outputs a polynomial $q(\cdot)$. After this point, the following steps will be iterated for $q(\lambda)$ times. In each iteration $y \in [q(\lambda)]$, some arbitrary circuit C_y ($|C_y| \leq L$) is computed.
3. **Simulating first round messages of honest parties**: The simulator evokes $\mathsf{Sim}_{\mathsf{MPC}}$ on inputs $\left(1^\lambda, \mathsf{F}\right)$, who first outputs the simulated first round messages of the honest parties: $\left\{\mathsf{msg}_i^{(1)}\right\}_{i\in H}$. Then Sim saves the state of $\mathsf{Sim}_{\mathsf{MPC}}$, and for each $i \in H$, sets $\mathsf{BFMS.msg}_i^{(1)} := \mathsf{msg}_i^{(1)}$ and sends it to $\mathcal{A}$ on behalf of honest parties.
4. **Receiving first round messages from corrupted parties**: The adversary outputs first round messages on behalf of corrupted parties: $\left\{\mathsf{BFMS.msg}_j^{(1}\right\}_{j\in I}$. The simulator then continues $\mathsf{Sim}_{\mathsf{MPC}}$ with these values as input. This step allows to extract corrupted party inputs $\{(x_j, r_j)\}_{j\in I}$ to the vanilla MPC. Sim then saves the state of $\mathsf{Sim}_{\mathsf{MPC}}$.
5. **Receiving output from ideal functionality**: Each honest party $i \in H$ sends its input x_i to the ideal functionality, and the simulator sends corrupted party inputs $\{x_j\}_{j\in I}$ on behalf of the adversary $\mathcal{A}$. The ideal functionality computes $Y_y \leftarrow C_y(x_1, \ldots, x_m)$ and sends Y_y to the simulator.
6. **Simulating second round messages of honest parties**:
   (a) The simulator evokes $\mathsf{Sim}_{\mathsf{NISC}}$ with input $\left(I, \{x_j\}_{j\in I}\right)$. Instead of letting it generate its own CRS^*, hardwires the value CRS^* to $\mathsf{Sim}_{\mathsf{NISC}}$ (However it can generate its own CRS' after the first iteration). By our previous construction, $\mathsf{Sim}_{\mathsf{NISC}}$ will first output $\{s_j\}_{j\in I}$ and the simulated commitments for honest parties: $\{c_i\}_{i\in H}$. Sim saves the state of $\mathsf{Sim}_{\mathsf{NISC}}$.
   (b) For each $j \in I$, Sim computes $c_j \leftarrow \mathsf{Commit}(j, x_j, s_j, \mathsf{CRS}')$. Sim then computes the circuit F' on inputs $\left(\{s_j\}_{j\in I}, \{c_j\}_{j\in[m]}, \{r_j\}_{j\in I}\right)$, and obtains $\left\{\widehat{K}_i\right\}_{i\in[m]}$.

(c) Sim continues to run $\mathsf{Sim_{MPC}}$: when it queries the ideal functionality, send $\left\{\widehat{K}_i\right\}_{i\in[m]}$ to it on behalf of the ideal functionality. $\mathsf{Sim_{MPC}}$ will then output the simulated second round messages for honest parties: $\left\{\mathsf{msg}_i^{(2)}\right\}_{i\in H}$.

(d) Sim continues to run $\mathsf{Sim_{NISC}}$, and sends $\{c_j\}_{j\in I}$ to it on behalf of corrupted parties. When it queries its own ideal functionality, Sim sends Y_y to it on behalf of the ideal functionality. $\mathsf{Sim_{NISC}}$ eventually outputs $\{m_i\}_{i\in H}$.

(e) For each $i \in H$, Sim does the following:
- Compute $\left(\widetilde{C}_i, \widehat{\mathbf{lab}}_i\right) \leftarrow \mathsf{Sim_G}\left(1^{|\mathsf{Compute}|},\ 1^n,\ m_i\right)$.
- For each $t \in [n]$, randomly sample another set of keys $S_{i,t} \leftarrow \{0,1\}^\lambda$; Then set $\overline{K}_i := \{(K_{i,t}, S_{i,t})\}_{t\in[n]}$.
- Set $\overline{\mathbf{lab}}_i := \left(\widehat{\mathbf{lab}}_i, \widehat{\mathbf{lab}}_i\right)$.
- Compute $\overline{\mathbf{elab}}_i \leftarrow \mathsf{LabEnc}\left(\overline{K}_i, \overline{\mathbf{lab}}_i\right)$.

(f) For each $i \in H$, set $\mathsf{BFMS.msg}_i^{(2)} := \left(\mathsf{msg}_i^{(2)}, \widetilde{C}_i, \overline{\mathbf{elab}}_i\right)$.

Indistinguishability Due to the reusability of large CRS (in particular, CRS^*) in the multiparty silent NISC, we only have to consider the case where a single circuit C is computed by the bounded FMS-MPC protocol (e.g. $q(\lambda) = 1$). We argue indistinguishability using the following hybrids:

Hybrid_0 : This is the real world execution.

Hybrid_1 : In this hybrid we make the following changes: for each $i \in H$, instead of setting $\overline{K}_i = \mathsf{PRF}\left(r_i, (t,b)\right)_{t\in[n], b\in\{0,1\}}$, sample all these keys uniformly at random. Indistinguishability follows from the security of PRF.

Hybrid_2 : In this hybrid, we simulate the underlying vanilla two-round MPC protocol.

1. Evoke $\mathsf{Sim_{MPC}}$ on inputs $\left(1^\lambda, \mathsf{F}\right)$, who first outputs the simulated first round messages of the honest parties: $\left\{\mathsf{msg}_i^{(1)}\right\}_{i\in H}$. Then save the state of $\mathsf{Sim_{MPC}}$, and for each $i \in H$, set $\mathsf{BFMS.msg}_i^{(1)} := \mathsf{msg}_i^{(1)}$ and send it to $\mathcal{A}$ on behalf of honest parties.
2. Receive $\left\{\mathsf{BFMS.msg}_j^{(1}\right\}_{j\in I}$ from $\mathcal{A}$.
3. Continue to run $\mathsf{Sim_{MPC}}$ with these values as input. Extract corrupted party inputs $\{(x_j, r_j)\}_{j\in I}$ to the vanilla MPC. Then save the state of $\mathsf{Sim_{MPC}}$.
4. When $\mathsf{Sim_{MPC}}$ queries the ideal functionality, proceed the following steps:
   - Using corrupted party inputs $\{x_j\}_{j\in I}$ and honest party inputs $\{x_i\}_{i\in H}$, compute step 1 and step 2 as described in circuit S honestly. Record the values $\{s_i\}_{i\in[m]}$ and $\{c_i\}_{i\in[m]}$.
   - Compute the circuit F' on inputs $\left(\{s_j\}_{j\in I}, \{c_j\}_{j\in[m]}, \{r_j\}_{j\in I}\right)$, and obtain $\left\{\widehat{K}_i\right\}_{i\in[m]}$.

   Send $\left\{\widehat{K}_i\right\}_{i\in[m]}$ to $\mathsf{Sim_{MPC}}$, who will then output the simulated second round messages for honest parties: $\left\{\mathsf{msg}_i^{(2)}\right\}_{i\in H}$.
5. For each $i \in H$, set $\mathsf{BFMS.msg}_i^{(2)} := \left(\mathsf{msg}_i^{(2)}, \widetilde{C}_i, \overline{\mathbf{elab}}_i\right)$, where $\widetilde{C}_i$ and $\overline{\mathbf{elab}}_i$ are still honestly computed (using random keys).

Indistinguishability follows directly from the security of vanilla two-round MPC protocol.

Hybrid_3 : In this hybrid, we change how honest party input labels are encrypted:

Let $z_i := \left(s_i, \{c_j\}_{j\in[m]}\right)$ where both values are obtained from previous hybrid. For $i \in H$, let $\widehat{\mathbf{lab}}_i$ be its input labels corresponding to the value z_i. Set $\overline{\mathbf{lab}}_i := \left(\widehat{\mathbf{lab}}_i, \widehat{\mathbf{lab}}_i\right)$. For each $t \in [n]$, if $z_i[t] = 0$, we encrypt the first input label using $K_{i,t}$, and the second input label using $S_{i,t}$. If $z_i[t] = 1$, switch the order of encryption. Indistinguishability follows from the semantic security of the label encryption scheme.

Hybrid_4 : In this hybrid, we make the following changes to how honest party input labels are encrypted:

We do not preserve the order of encryption. Instead, for $i \in H$, set $\overline{K}_i := \{(K_{i,t}, S_{i,t})\}_{t\in[n]}$ and compute $\overline{\mathbf{elab}}_i \leftarrow \mathsf{LabEnc}\left(\overline{K}_i, \overline{\mathbf{lab}}_i\right)$. Indistinguishability follows from the fact that the label encryption scheme always randomly permutes the ciphertexts.

Hybrid_5 : In this hybrid, we simulate honest party garbled circuits:

1. For each $i \in H$, honestly compute $m_i \leftarrow \mathsf{Compute}(i, x_i, s_i, \mathsf{CRS}^*, \{c_j\}_{j\in[m]}, C)$, where s_i and $\{c_j\}_{j\in[m]}$ are still obtained from Hybrid_2.
2. For each $i \in H$, simulate its garbled circuit: $\left(\widetilde{C}_i, \widehat{\mathbf{lab}}_i\right) \leftarrow \mathsf{Sim}_\mathsf{G}\left(1^{|\mathsf{Compute}|},\ 1^n,\ m_i\right)$.
3. Set $\overline{\mathbf{lab}}_i := \left(\widehat{\mathbf{lab}}_i, \widehat{\mathbf{lab}}_i\right)$.
4. Compute $\overline{\mathbf{elab}}_i \leftarrow \mathsf{LabEnc}\left(\overline{K}_i, \overline{\mathbf{lab}}_i\right)$.

Indistinguishability directly follows from the security of garbling scheme.

Hybrid_6 : In the last hybrid, we simulate the underlying multiparty silent NISC protocol:

1. Evoke $\mathsf{Sim}_\mathsf{NISC}$ with input $\left(I, \{x_j\}_{j\in I}\right)$. By our previous construction, $\mathsf{Sim}_\mathsf{NISC}$ will generate its own random CRS. We include this CRS in the view that Sim outputs. Then $\mathsf{Sim}_\mathsf{NISC}$ will output $\{s_j\}_{j\in I}$ and $\{c_i\}_{i\in H}$.
2. For each $j \in I$, using extracted corrupted party inputs $\{x_j\}_{j\in I}$, compute $c_j \leftarrow \mathsf{Commit}(j, x_j, s_j, \mathsf{CRS}')$. Send $\{c_j\}_{j\in I}$ to $\mathsf{Sim}_\mathsf{NISC}$ on behalf of the adversary $\mathcal{A}$. By our construction, $\mathsf{Sim}_\mathsf{NISC}$ will then query its ideal functionality.
3. Query the ideal functionality and obtain $Y \leftarrow C\,(x_1, \ldots, x_m)$.
4. Send Y to $\mathsf{Sim}_\mathsf{NISC}$ on behalf of the ideal functionality, and $\mathsf{Sim}_\mathsf{NISC}$ responds with $\{m_i\}_{i\in H}$.
5. Moreover, when $\mathsf{Sim}_\mathsf{MPC}$ queries the ideal functionality, compute circuit F' directly with inputs $\left(\{s_j\}_{j\in I}, \{c\}_{j\in[m]}, \{r_j\}_{j\in I}\right)$, and give this output to $\mathsf{Sim}_\mathsf{MPC}$. Similarly, when simulating honest party garbled circuits, use $\{m_i\}_{i\in H}$ instead.

Indistinguishability follows from the security of multiparty silent NISC protocol.

G Security of FMS-MPC

We prove malicious security of this FMS-MPC protocol, relying on the following ingredients: Let $\mathsf{Sim}_\mathsf{BFMS}$ be the simulator for a maliciously secure bounded FMS-MPC protocol, $\mathsf{Sim}_\mathsf{CRE}$ be the simulator for randomized encoding scheme, and Sim_G be a simulator for garbling scheme. Let the adversary $\mathcal{A}$ corrupt a set of parties $I \subset [m]$ in a static manner.

Description Of Simulator The ideal world adversary Sim simulates the view of real world adversary as follows. It makes use the circuit E' defined below.

1. **Sampling parameters**: The simulator sets $L = \max(|\mathsf{E}|, |\mathsf{F}_y|)$, and evokes $\mathsf{Sim}_\mathsf{BFMS}$ on inputs $(1^\lambda, I, L)$. Denote this simulator (in this iteration) by $\mathsf{Sim}^{(1,1)}_\mathsf{BFMS}$. It first outputs the CRS. Sim forwards this CRS to $\mathcal{A}$ and saves the state of $\mathsf{Sim}^{(1,1)}_\mathsf{BFMS}$.
2. **Receiving a circuit description from the adversary**: The adversary specifies a circuit C, whose size can be an arbitrary polynomial in λ. Let $|C| = q(\lambda)$. $\mathcal{A}$ gives the description of C to Sim, who then forwards $q(\cdot)$ and E to $\mathsf{Sim}^{(1,1)}_\mathsf{BFMS}$.
3. **Simulating first round messages of honest parties**: Sim continues to run $\mathsf{Sim}^{(1,1)}_\mathsf{BFMS}$, who then outputs the simulated first round message of honest parties: $\left\{\mathsf{msg}^{(1)}_i\right\}_{i\in H}$. For each $i \in H$, Sim sets $\mathsf{FMS.msg}^{(1)}_i := \mathsf{msg}^{(1)}_i$ and gives it to $\mathcal{A}$ on behalf of honest parties. Then it saves the state of $\mathsf{Sim}^{(1,1)}_\mathsf{BFMS}$.

4. **Receiving first round messages from corrupted parties**: The adversary outputs first round messages on behalf of corrupted parties: $\left\{\mathsf{FMS.msg}_j^{(1}\right\}_{j\in I}$. Sim continues $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$ with these values as input. This step allows to extract corrupted party inputs $\{((x_j, v_j), r_j)\}_{j\in I}$ to the bounded FMS-MPC. Sim then saves the state of $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$.
5. **Receiving output from ideal functionality**: Each honest party $i \in H$ sends its input x_i to the ideal functionality, and the simulator sends corrupted party inputs $\{x_j\}_{j\in I}$ on behalf of the adversary $\mathcal{A}$. The ideal functionality computes $Y \leftarrow C(x_1, \ldots, x_m)$ and sends Y to the simulator.
6. **Simulating second round messages of honest parties**: The rest of simulation proceeds as follows:
   - **Constructing the binary tree**: Sim computes $[G_y]_{y\in[n]} \leftarrow \mathsf{CRE.Enc}\left(1^\lambda, C\right)$ (where $n = O(q(\lambda))$). Then it constructs a complete binary tree according to protocol specification. We adopt the following notations as before: let d be the depth of this binary tree, and let the index tuple (k, y) denote the y^{th} node at the k^{th} level in this tree.
   - **Simulating root node**: Sim continues to run $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$ until it queries the ideal functionality. At this point, the simulator does the following:
     (a) For each $j \in I$, compute $r_j^{(2,1)} := \mathsf{G}_0(r_j)$, $r_j^{(2,2)} := \mathsf{G}_1(r_j)$.
     (b) Start two new iterations of $\mathsf{Sim}_{\mathsf{BFMS}}$ and denote them by $\mathsf{Sim}_{\mathsf{BFMS}}^{(2,1)}$ and $\mathsf{Sim}_{\mathsf{BFMS}}^{(2,2)}$ respectively. Give the initial inputs $\left(1^\lambda, \mathsf{E}, \left\{(x_j, v_j), r_j^{(2,1)}\right\}_{j\in I}\right)$ to the former, and $\left(1^\lambda, \mathsf{E}, \left\{(x_j, v_j), r_j^{(2,2)}\right\}_{j\in I}\right)$ to the latter.
     (c) $\mathsf{Sim}_{\mathsf{BFMS}}^{(2,1)}$ and $\mathsf{Sim}_{\mathsf{BFMS}}^{(2,2)}$ output simulated first round messages for honest parties: $\left\{\mathsf{msg}_i^{(1),(2,1)}\right\}_{i\in H}$ and $\left\{\mathsf{msg}_i^{(1),(2,2)}\right\}_{i\in H}$. Then save their states.
     (d) Sample two fresh $\left(\mathsf{CRS'}^{(2,1)}, \mathsf{CRS'}^{(2,2)}\right) \leftarrow \mathsf{Gen}(1^\lambda)$ and then generate the states and first round messages on behalf of corrupted parties: for each $j \in I$, compute
     $$\left(\mathsf{st}_j^{(1),(2,1)}, \mathsf{msg}_j^{(1),(2,1)}\right) \leftarrow \mathsf{MPC}_1\left(1^\lambda, \mathsf{CRS'}^{(2,1)}, j, ((x_j, v_j), r_j^{(2,1)})\right),$$
     $$\left(\mathsf{st}_j^{(1),(2,2)}, \mathsf{msg}_j^{(1),(2,2)}\right) \leftarrow \mathsf{MPC}_1\left(1^\lambda, \mathsf{CRS'}^{(2,2)}, j, ((x_j, v_j), r_j^{(2,2)})\right).$$
     (e) Compute circuit E' on input: $\left(\{r_j\}_{j\in I}, \left\{\left(\mathsf{msg}_j^{(1),(2,1)},\ \mathsf{msg}_j^{(1),(2,2)}\right)\right\}_{j\in[m]}, \left\{\left(\mathsf{st}_j^{(1),(2,1)}, \mathsf{st}_j^{(1),(2,2)}\right)\right\}_{j\in I}, \left(\mathsf{CRS'}^{(2,1)}, \mathsf{CRS'}^{(2,2)}\right)\right)$. Obtain $\left(\left\{\widehat{K}_i^{(2,1)} := \widehat{K}_i^0\right\}_{i\in[m]}, \left\{\widehat{K}_i^{(2,2)} := \widehat{K}_i^1\right\}_{i\in[m]}\right)$ and forward them to $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$ on behalf of the ideal functionality.
     (f) $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$ outputs the simulated second round messages for honest parties: $\left\{\mathsf{msg}_i^{(2)}\right\}_{i\in H}$.
   - **Simulating all other nodes**: For each $k \in [2, d]$, the simulator runs the subroutine Sim^k (described in algorithm $G.2$ and $G.3$). It will simulate honest party garbled circuits and their label encryption for all node computation at level k. Sim^k will also output the inputs of Sim^{k+1} so that these subroutines can be run in sequential.

Algorithm G.1 (Circuit E').

Input: $\left(\{r_j\}_{j\in I}, \left\{\left(\mathsf{msg}_j^{(1),0},\ \mathsf{msg}_j^{(1),1}\right)\right\}_{j\in[m]},\ \left\{\left(\mathsf{st}_j^{(1),0}, \mathsf{st}_j^{(1),1}\right)\right\}_{j\in I}, \left(\mathsf{CRS}'^{\,0}, \mathsf{CRS}'^{\,1}\right)\right)$.

1. *For each $i \in [1, m]$ (Generating the left child):*
   - *If $i \in I$, then:*
     (a) *Set* $k_i^0 := \mathsf{H}_0(r_i)$, $z_i^0 := \left(\mathsf{CRS}'^{\,0}, \mathsf{st}_i^{(1),0}, \left\{\mathsf{msg}_j^{(1),0}\right\}_{j\in[m]}\right)$.
     (b) *Let* $l := |z_i^0|$. *For* $t \in [l]$, *set* $K_{i,t}^0 := \mathsf{PRF}\left(k_i^0, (t, z_i^0[t])\right)$.
     (c) *Set* $\widehat{K}_i^0 := \left\{K_{i,t}^0\right\}_{t\in[l]}$.
   - *If $i \in H$, then:*
     (a) *For each* $t \in [l]$, *randomly sample* $K_{i,t}^0 \leftarrow \{0,1\}^\lambda$;
     (b) *Set* $\widehat{K}_i^0 = \{K_{i,t}^0\}_{t\in[l]}$.
2. *For each $i \in [m]$ (Generating the right child):*
   - *If $i \in I$, then:*
     (a) *Set* $k_i^1 := \mathsf{H}_1(r_i)$, $z_i^1 := \left(\mathsf{CRS}'^{\,1}, \mathsf{st}_i^{(1),1}, \left\{\mathsf{msg}_j^{(1),1}\right\}_{j\in[m]}\right)$.
     (b) *Let* $l := |z_i^1|$. *For* $t \in [l]$, *set* $K_{i,t}^1 := \mathsf{PRF}\left(k_i^1, (t, z_i^1[t])\right)$.
     (c) *Set* $\widehat{K}_i^1 := \left\{K_{i,t}^1\right\}_{t\in[l]}$.
   - *If $i \in H$, then:*
     (a) *For each* $t \in [l]$, *randomly sample* $K_{i,t}^1 \leftarrow \{0,1\}^\lambda$;
     (b) *Set* $\widehat{K}_i^1 = \{K_{i,t}^1\}_{t\in[l]}$.
3. *Output* $\left\{\widehat{K}_i^0\right\}_{i\in[m]}$ *and* $\left\{\widehat{K}_i^1\right\}_{i\in[m]}$.

Algorithm G.2 ($\mathsf{Sim}^{k\in[2,d-1]}$)**.**

Input: $\left\{\left\{r_j^{(k,y)}\right\}_{j\in I},\ \left\{\mathsf{msg}_j^{(1),(k,y)}\right\}_{j\in I},\ \left\{\widehat{K}_i^{(k,y)}\right\}_{i\in H}\right\}_{y\in[2^{k-1}]}$.

For each $y\in[2^{k-1}]$*:*

1. *Continue to run* $\mathsf{Sim}_{\mathsf{BFMS}}^{k,y}$*. Send to it values* $\left\{\mathsf{msg}_j^{(1),(k,y)}\right\}_{j\in I}$ *on behalf of corrupted parties. Wait until it queries the ideal functionality and then save its state.*
2. *For each* $j\in I$*, compute* $r_j^{(k+1,2y-1)} := \mathsf{G}_0\left(r_j^{(k,y)}\right),\ r_j^{(k+1,2y)} := \mathsf{G}_1\left(r_j^{(k,y)}\right)$.
3. *Start two new iterations of* $\mathsf{Sim}_{\mathsf{BFMS}}$ *and denote them by* $\mathsf{Sim}_{\mathsf{BFMS}}^{(k+1,2y-1)}$ *and* $\mathsf{Sim}_{\mathsf{BFMS}}^{(k+1,2y)}$ *respectively. Give the initial inputs* $\left(1^\lambda,\mathsf{E},\left\{(x_j,v_j),r_j^{(k+1,2y-1)}\right\}_{j\in I}\right)$ *to the former, and* $\left(1^\lambda,\mathsf{E},\left\{(x_j,v_j),r_j^{(k+1,2y)}\right\}_{j\in I}\right)$ *to the latter.*
4. $\mathsf{Sim}_{\mathsf{BFMS}}^{(k+1,2y-1)}$ *and* $\mathsf{Sim}_{\mathsf{BFMS}}^{(k+1,2y)}$ *output simulated first round messages for honest parties:* $\left\{\mathsf{msg}_i^{(1),(k+1,2y-1)}\right\}_{i\in H}$ *and* $\left\{\mathsf{msg}_i^{(1),(k+1,2y)}\right\}_{i\in H}$*. Then save their states.*
5. *Sample two fresh* $\left(\mathsf{CRS'}^{(k+1,2y-1)},\ \mathsf{CRS'}^{(k+1,2y)}\right)\leftarrow\mathsf{Gen}(1^\lambda)$.
6. *Generate the states and first round messages twice on behalf of corrupted parties:*
   For each $j\in I$*, compute:*
   $\left(\mathsf{st}_j^{(1),(k+1,2y-1)},\mathsf{msg}_j^{(1),(k+1,2y-1)}\right)\leftarrow\mathsf{MPC}_1\left(1^\lambda,\mathsf{CRS'}^{(k+1,2y-1)},j,((x_j,v_j),r_j^{(k+1,2y-1)})\right)$;
   $\left(\mathsf{st}_j^{(1),(k+1,2y)},\mathsf{msg}_j^{(1),(k+1,2y)}\right)\leftarrow\mathsf{MPC}_1\left(1^\lambda,\mathsf{CRS'}^{(k+1,2y)},j,((x_j,v_j),r_j^{(k+1,2y)})\right)$.
7. *Compute circuit* E' *on inputs* $\left\{r_j^{(k,y)}\right\}_{j\in I},\left\{\left(\mathsf{msg}_j^{(1),(k+1,2y-1)},\ \mathsf{msg}_j^{(1),(k+1,2y)}\right)\right\}_{j\in[m]}$ $,\left\{\left(\mathsf{st}_j^{(1),(k+1,2y-1)},\mathsf{st}_j^{(1),(k+1,2y)}\right)\right\}_{j\in I}$ *and* $\left(\mathsf{CRS'}^{(k+1,2y-1)},\ \mathsf{CRS'}^{(k+1,2y)}\right)$.
   Obtain $\left(\left\{\widehat{K}_i^{(k+1,2y-1)}:=\widehat{K}_i^0\right\}_{i\in[m]},\ \left\{\widehat{K}_i^{(k+1,2y)}:=\widehat{K}_i^1\right\}_{i\in[m]}\right)$ *and forward them to* $\mathsf{Sim}_{\mathsf{BFMS}}^{(k,y)}$ *on behalf of the ideal functionality.*
8. *Continue to run* $\mathsf{Sim}_{\mathsf{BFMS}}^{(k,y)}$*, which outputs the simulated second round messages for honest parties:* $\left\{\mathsf{msg}_i^{(2),(k,y)}\right\}_{i\in H}$.
9. *For each* $i\in H$*:*
   (a) *Compute* $\left(\widetilde{C}_i^{(k,y)},\widehat{\boldsymbol{lab}}_i^{(k,y)}\right)\leftarrow\mathsf{Sim}_{\mathsf{G}}\left(1^{|\mathsf{MPC}_2|},1^l,\left(\mathsf{msg}_i^{(2),(k,y)}\right)\right)$.
   (b) *For* $t\in[l]$*, randomly sample another set of keys:* $S_{i,t}^{(k,y)}\leftarrow\{0,1\}^\lambda$*. Set* $\widehat{S}_i^{(k,y)} := \left\{S_{i,t}^{(k,y)}\right\}_{t\in[l]}$ *and* $\overline{K}_i^{(k,y)} := \left(\widehat{K}_i^{(k,y)},\ \widehat{S}_i^{(k,y)}\right)$.
   (c) *Set* $\overline{\boldsymbol{lab}}_i^{(k,y)} := \left(\widehat{\boldsymbol{lab}}_i^{(k,y)},\widehat{\boldsymbol{lab}}_i^{(k,y)}\right)$.
   (d) *Compute* $\overline{\boldsymbol{elab}}_i^{(k,y)}\leftarrow\mathsf{LabEnc}\left(\overline{\boldsymbol{lab}}_i^{(k,y)},\overline{K}_i^{(k,y)}\right)$.
10. *Output* $\left\{\widetilde{C}_i^{(k,y)},\ \overline{\boldsymbol{elab}}_i^{(k,y)}\right\}_{i\in H}$.
11. *Output* $\left(\left\{r_j^{(k+1,2y-1)}\right\}_{j\in I},\left\{\mathsf{msg}_j^{(1),(k+1,2y-1)}\right\}_{j\in I},\left\{\widehat{K}_i^{(k+1,2y-1)}\right\}_{i\in H}\right)$ *and* $\left(\left\{r_j^{(k+1,2y)}\right\}_{j\in I},\left\{\mathsf{msg}_j^{(1),(k+1,2y)}\right\}_{j\in I},\left\{\widehat{K}_i^{(k+1,2y)}\right\}_{i\in H}\right)$.

Algorithm G.3 (Sim^d)**.**

Input: $\left\{\left\{\mathsf{msg}_j^{(1),(d,y)}\right\}_{j\in I}, \left\{\widehat{K}_i^{(d,y)}\right\}_{i\in H}\right\}_{y\in[n]}$.

1. *Simulate randomized encodings:* $\left[\widetilde{G}_y\right]_{y\in[n]} \leftarrow \mathsf{Sim}_{\mathsf{CRE}}\left(1^\lambda, C, Y\right)$.
2. *For each* $y \in [n]$*:*
   (a) *Continue to run* $\mathsf{Sim}_{\mathsf{BFMS}}^{d,y}$. *Send to it values* $\left\{\mathsf{msg}_j^{(1),(d,y)}\right\}_{j\in I}$ *on behalf of corrupted parties. Wait until it queries the ideal functionality.*
   (b) *Send* $\widetilde{G}_y$ *to* $\mathsf{Sim}_{\mathsf{BFMS}}^{(d,y)}$ *on behalf of the ideal functionality.*
   (c) *Proceed step 8, 9 and 10 as the previous simulator.*

Indistinguishability We argue indistinguishability using the following hybrids:

Hybrid_0 : This hybrid is the same as real world execution.

Hybrid_1 : In this hybrid, we make the following changes to honest party secret randomness and PRF keys: for each $i \in H$, instead of setting:

$$r_i^{(2,1)} := \mathsf{G}_0\left(r_i^{(1,1)}\right),\ r_i^{(2,2)} := \mathsf{G}_1\left(r_i^{(1,1)}\right); k_i^{(2,1)} := \mathsf{H}_0\left(r_i^{(1,1)}\right),\ k_i^{(2,2)} := \mathsf{H}_1\left(r_i^{(1,1)}\right).$$

Sample them uniformly at random. Indistinguishability follows from the security of PRG.

Hybrid_2 : In this hybrid we make the following changes to honest party label encryption keys: for each $i \in H$, instead of setting

$$\overline{K}_i^{(2,1)} = \mathsf{PRF}\left(k_i^{(2,1)}, (t,b)\right)_{t\in[l],b\in\{0,1\}},\ \overline{K}_i^{(2,2)} = \mathsf{PRF}\left(k_i^{(2,2)}, (t,b)\right)_{t\in[l],b\in\{0,1\}}.$$

Sample all those keys uniformly at random. Indistinguishability follows from the security of PRF.

Hybrid_3 : In this hybrid, we simulate the root node computation.

1. Evoke $\mathsf{Sim}_{\mathsf{BFMS}}$ on inputs $\left(1^\lambda, I, L, |C|\right)$. Denote this simulator (in this iteration) by $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$. It first outputs the CRS. Broadcast CRS and save the state of $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$.
2. Give E to $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$, who then outputs the simulated first round message of honest parties: $\left\{\mathsf{msg}_i^{(1)}\right\}_{i\in H}$. For each $i \in H$, set $\mathsf{FMS.msg}_i^{(1)} := \mathsf{msg}_i^{(1)}$ and forward them to corrupted parties on behalf of honest parties.
3. The corrupted parties output their first round messages: $\left\{\mathsf{FMS.msg}_j^{(1}\right\}_{j\in I}$. Give them to $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$ so as to extract corrupted party inputs $\{((x_j, v_j), r_j)\}_{j\in I}$.
4. Wait until $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$ queries the ideal functionality. At this point, using honest party and corrupted party inputs $\{(x_j, v_j)\}_{j\in[m]}$ and the values $\left\{(r_i^{(2,1)}, r_i^{(2,2)}, k_i^{(2,1)}, k_i^{(2,2)})\right\}_{i\in H}$, honestly compute the circuit E (while ignoring PRG and PRF) and give this output to $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$. Moreover, for $y \in [1,2]$, record the values $\mathsf{CRS}'^{\,(2,y)}$, $\left\{\mathsf{st}_i^{(1),(2,y)}\right\}_{i\in H}$ and $\left\{\mathsf{msg}_j^{(1),(2,y)}\right\}_{j\in[m]}$.
5. $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$ outputs the simulated second round messages for honest parties. This completes our simulation for the root node computation.

Indistinguishability follows from the security of bounded FMS-MPC protocol.

Hybrid_4 : In this hybrid, we change how honest party input labels are encrypted at level 2:

For $i \in H$ and for $y \in [1,2]$, let $\widehat{\mathbf{lab}}_i^{(2,y)}$ be its input labels corresponding to the value

$$z_i^{(2,y)} := \left(\mathsf{CRS}'^{\,(2,y)}, \mathsf{st}_i^{(1),(2,y)}, \left\{ \mathsf{msg}_j^{(1),(2,y)} \right\}_{j \in [m]} \right).$$

Set $\overline{\mathbf{lab}}_i^{(2,y)} := \left(\widehat{\mathbf{lab}}_i^{(2,y)}, \widehat{\mathbf{lab}}_i^{(2,y)} \right)$. For each $t \in [l]$, if $z_i^{(2,y)}[t] = 0$, we encrypt the first input label using $K_{i,t}^{(2,y)}$, and the second input label using $S_{i,t}^{(2,y)}$. If $z_i^{(2,y)}[t] = 1$, switch the order of encryption. Indistinguishability follows from the semantic security of the label encryption scheme.

Hybrid_5 : In this hybrid, we again change how honest party input labels are encrypted:

We do not preserve the order of encryption. Instead, for $i \in H$ and for $y \in [1,2]$, set $\overline{K}_i^{(2,y)} := \left\{ \left(K_{i,t}^{(2,y)}, S_{i,t}^{(2,y)} \right) \right\}_{t \in [l]}$ and compute $\overline{\mathbf{elab}}_i^{(2,y)} \leftarrow \mathsf{LabEnc}\left(\overline{K}_i^{(2,y)}, \overline{\mathbf{lab}}_i^{(2,y)} \right)$. Indistinguishability follows from the fact that the label encryption scheme always randomly permutes the ciphertexts.

Hybrid_6 : In this hybrid, we simulate honest party garbled circuits at level 2: For each $i \in H$ for $y \in [1,2]$:

1. Honestly compute $\mathsf{msg}_i^{(2),(2,y)} \leftarrow \mathsf{MPC}_2\left(\mathsf{E}, (\mathsf{CRS}'^{\,(2,y)}, \mathsf{CRS}^*), \mathsf{st}_i^{(1),(2,y)}, \left\{ \mathsf{msg}_j^{(1),(2,y)} \right\}_{j \in [m]} \right)$, where $\mathsf{CRS}'^{\,(2,y)}$, $\mathsf{st}_i^{(1),(2,y)}$ and $\left\{ \mathsf{msg}_j^{(1),(2,y)} \right\}_{j \in [m]}$ are the same as Hybrid_3.
2. Simulate its garbled circuit: $\left(\widetilde{C}_i^{(2,y)}, \widehat{\mathbf{lab}}_i^{(2,y)} \right) \leftarrow \mathsf{Sim}_{\mathsf{G}}\left(1^{|\mathsf{MPC}_2|},\ 1^l,\ \mathsf{msg}_i^{(2),(2,y)} \right)$.
3. Set $\overline{\mathbf{lab}}_i^{(2,y)} := \left(\widehat{\mathbf{lab}}_i^{(2,y)}, \widehat{\mathbf{lab}}_i^{(2,y)} \right)$.
4. Compute $\overline{\mathbf{elab}}_i^{(2,y)} \leftarrow \mathsf{LabEnc}\left(\overline{K}_i^{(2,y)}, \overline{\mathbf{lab}}_i^{(2,y)} \right)$.

Indistinguishability follows from the security of garbling scheme.

Hybrid_7 : In this hybrid we simulate all node computations at level 2 (Part I):

We begin with a slight modification on Hybrid_3: when $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$ queries the ideal functionality, instead of computing the expansion circuit E honestly, we do the following:

1. Start two new iterations of $\mathsf{Sim}_{\mathsf{BFMS}}$ and denote them by $\mathsf{Sim}_{\mathsf{BFMS}}^{(2,1)}$ and $\mathsf{Sim}_{\mathsf{BFMS}}^{(2,2)}$ respectively. Give the initial inputs $\left(1^\lambda, \mathsf{E}, \left\{ (x_j, v_j), r_j^{(2,1)} \right\}_{j \in I} \right)$ to the former, and $\left(1^\lambda, \mathsf{E}, \left\{ (x_j, v_j), r_j^{(2,2)} \right\}_{j \in I} \right)$ to the latter.
2. $\mathsf{Sim}_{\mathsf{BFMS}}^{(2,1)}$ and $\mathsf{Sim}_{\mathsf{BFMS}}^{(2,2)}$ output simulated first round messages for honest parties: $\left\{ \mathsf{msg}_i^{(1),(2,1)} \right\}_{i \in H}$ and $\left\{ \mathsf{msg}_i^{(1),(2,2)} \right\}_{i \in H}$. Save their states.
3. Generate the states and first round messages on behalf of corrupted parties:
   For each $j \in I$, compute $\left(\mathsf{st}_j^{(1),(2,1)}, \mathsf{msg}_j^{(1),(2,1)} \right) \leftarrow \mathsf{MPC}_1\left(1^\lambda, \mathsf{CRS}'^{\,(2,1)}, j, ((x_j, v_j), r_j^{(2,1)}) \right)$;
   $\left(\mathsf{st}_j^{(1),(2,2)}, \mathsf{msg}_j^{(1),(2,2)} \right) \leftarrow \mathsf{MPC}_1\left(1^\lambda, \mathsf{CRS}'^{\,(2,2)}, j, ((x_j, v_j), r_j^{(2,2)}) \right)$.
4. Compute circuit E' on input:
   $$\left(\{r_j\}_{j \in I}, \left\{ \left(\mathsf{msg}_j^{(1),(2,1)},\ \mathsf{msg}_j^{(1),(2,2)} \right) \right\}_{j \in [m]},\ \left\{ \left(\mathsf{st}_j^{(1),(2,1)}, \mathsf{st}_j^{(1),(2,2)} \right) \right\}_{j \in I}, \left(\mathsf{CRS}'^{\,(2,1)}, \mathsf{CRS}'^{\,(2,2)} \right) \right).$$
   Obtain
   $$\left(\left\{ \widehat{K}_i^{(2,1)} \right\}_{i \in [m]},\ \left\{ \widehat{K}_i^{(2,2)} \right\}_{i \in [m]} \right)$$
   and forward them to $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$ on behalf of the ideal functionality.
5. $\mathsf{Sim}_{\mathsf{BFMS}}^{(1,1)}$ outputs the simulated second round messages for honest parties: $\left\{ \mathsf{msg}_i^{(2)} \right\}_{i \in H}$.

Indistinguishability follows from the security of the bounded FMS-MPC.

Hybrid_8 : In this hybrid we simulate all node computations at level 2 (Part II):

For each $y \in [1,2]$, send $\left\{\mathsf{msg}_j^{(1),(2,y)}\right\}_{j\in I}$ to $\mathsf{Sim}_{\mathsf{BFMS}}^{(2,y)}$ and continue to run it. When it queries the ideal functionality, use the honest party inputs $\{(x_i, v_i)\}_{i\in H}$ and corrupted party inputs $\left\{\left((x_j, v_j), r_j^{(2,y)}\right)\right\}_{j\in I}$ to honestly compute the circuit E (while ignoring PRF and PRG just as before). Give this output to $\mathsf{Sim}_{\mathsf{BFMS}}^{(2,y)}$ on behalf of the ideal functionality. $\mathsf{Sim}_{\mathsf{BFMS}}^{(2,y)}$ will output the simulated second round messages for honest parties $\left\{\mathsf{msg}_i^{(2),(2,y)}\right\}_{i\in H}$. This completes our simulation for node computation at level 2.

Indistinguishability follows from the security of the bounded FMS-MPC. Besides, notice that this hybrid realizes Sim^2, except that all nodes at the next level are still honestly computed.

$\mathsf{Hybrid}_{k\in[9,d+5]}$: In this sequence of hybrids, let $t := k-4$, each Hybrid_k differs from Hybrid_{k-1} by executing the simulator Sim^t. Indistinguishability follows from the same arguments above.

Hybrid_{d+6} : In the last hybrid, we simulate the randomized encodings. First, query the ideal functionality with corrupted party inputs. The trusted party returns an output $Y \leftarrow C(x_1, \ldots, x_m)$. Evoke the simulator for randomized encoding scheme and compute $\left[\widetilde{G}_y\right]_{y\in[n]} \leftarrow \mathsf{Sim}_{\mathsf{CRE}}\left(1^\lambda, C, Y\right)$. Then for each $y \in [n]$, do the following:

Continue to run $\mathsf{Sim}_{\mathsf{BFMS}}^{d,y}$. When it queries the ideal functionality, send to it $\widetilde{G}_y$ on behalf of the ideal functionality. It will output the simulated second round messages for honest parties: $\left\{\mathsf{msg}_i^{(2),(d,y)}\right\}_{i\in H}$. This completes our simulation of the entire FMS-MPC protocol. Indistinguishability follows from the security of randomized encoding scheme. Finally, notice that this hybrid is the same as the ideal world execution.