Reusable Two-Round MPC from DDH

James Bartusek[1,⋆], Sanjam Garg[1,⋆], Daniel Masny[2], and Pratyay Mukherjee[2]

[1] University of California, Berkeley, USA
{jamesbartusek,sanjamg}@berkeley.edu
[2] Visa Research, Palo Alto, USA
{dmasny,pratmukh}@visa.com

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

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

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

1 Introduction

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

⋆ Supported in part from AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, DARPA SIEVE Award, 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 funding agencies.

so? Furthermore, what if they want to additionally compute circuits $\mathsf{C}_2, \mathsf{C}_3 \ldots$ at a later point on the same set of inputs?

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

Making progress towards this goal, Garg et al. [GGHR14] gave the first constructions of *two-round* MPC protocols, assuming indistinguishability obfuscation [GGH$^+$13] (or, witness encryption [GLS15,GGSW13]) and one-way functions.[3] A very nice feature of the Garg et al. construction is that the first round message is indeed reusable across multiple executions, thereby achieving the above vision. Follow up works realized two-round MPC protocols based on significantly weaker computational assumptions. In particular, two-round MPC protocols based on LWE were obtained [MW16,BP16,PS16], followed by a protocol based on bilinear maps [GS17,BF01,Jou04]. Finally, this line of work culminated with the recent works of Benmahouda and Lin [BL18] and Garg and Srinivasan [GS18], who gave constructions based on the minimal assumption that two-round oblivious transfer (OT) exists.

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

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

1.1 Our Result

In this work, we answer the above question by presenting a general compiler that obtains reusable two-round MPC, starting from any two-round MPC and

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

using homomorphic secret sharing (HSS) [BGI16] and pseudorandom functions in NC^1. In a bit more detail, our main theorem is:

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

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

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

Corollary 2. *Let $\mathcal{X} \in \{$semi-honest in plain model, malicious in common random/reference sting model$\}$. Assuming DDH, there exists a* reusable *two-round $\mathcal{X}$-MPC protocol.*

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

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

First Message Succinct Two-Round MPC. At the heart of this work is a construction of a *first message succinct* (FMS) two-round MPC protocol— that is, a two-round MPC protocol where the first message of the protocol is computed independently of the circuit being evaluated. In particular, the parties do not need to know the description of the circuit that will eventually be computed over their inputs in the second round. Furthermore, parties do not even need to know the *size* of the circuit to be computed in the second round.[4] This allows parties to publish their first round messages and later compute any arbitrary circuit on their inputs. Formally, we show the following:

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

Theorem 3. *Let $\mathcal{X} \in \{$semi-honest in plain model, malicious in common random/reference sting model$\}$. Assuming DDH, there exists a first message succinct two-round $\mathcal{X}$-MPC protocol.*

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

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

Theorem 4. *Let $\mathcal{X} \in \{$semi-honest in plain model, malicious in common random/reference sting model$\}$. Assuming a first message succinct two-round MPC protocol, there exists a* reusable *two-round $\mathcal{X}$-MPC protocol.*

Concurrent and Independent Work. Two recent independent works have also explicitly studied reusable two-round MPC, obtaining a variety of results. First, Benhamouda and Lin [BL20] construct reusable two-round MPC from assumptions on bilinear maps. Their techniques are quite different than those used in this paper and, while they need stronger assumptions than us, their protocol does have the advantage that the number of parties participating in the second round need not be known when generating first round messages. In our protocol, the number of parties in the system is a parameter used to generate the first round messages. Second, Ananth et al. [AJJ20] construct semi-honest reusable two-round MPC from lattices in the plain model. Prior work from lattices [MW16] required a CRS even in the semi-honest setting. The work of Ananth et al. [AJJ20] includes essentially the same transformation from "first message succinct" MPC to reusable MPC that constitutes the third step of our construction (see Section 2.3).

2 Technical Overview

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

2.1 Step 1: Overview of the scHSS Construction

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

The approach: A sharing-compact HSS scheme consists of three algorithms, Share, Eval, and Dec. Our construction follows the compiler of [GS18] that takes an arbitrary MPC protocol and squishes it to two rounds. At a high level, to share a secret x among n parties, we have the Share algorithm first compute an n-party additive secret sharing $x_1, \ldots, x_n$ of x. Then, it runs the first round of the squished n-party protocol on behalf of each party j with input x_j.[5] Finally, it sets the j'th share to be all of the first round messages, plus the secret state of the j'th party. The Eval algorithm run by party j will simply run the second round of the MPC, and output the resulting message. The Dec algorithm takes all second round messages and reconstructs the output.

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

The [GS18] compiler: Before further discussing the size dependence issue, we recall the [GS18] compiler. The compiler is applied to any *conforming* MPC protocol, a notion defined in [GS18].[6] Roughly, a conforming protocol operates via a sequence of actions $\phi_1, \ldots, \phi_T$. At the beginning of the protocol, each party j broadcasts a one-time pad of their input, and additionally generates some secret state v_j. The encrypted inputs are arranged into a global public state st, which will be updated throughout the protocol. At each step t, the action $\phi_t = (j, f, g, h)$ is carried out by having party j broadcast the bit $\gamma_t :=$

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

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

$\mathsf{NAND}(\mathsf{st}_f \oplus v_{j,f}, \mathsf{st}_g \oplus v_{j,g}) \oplus v_{j,h}$. Everybody then updates the global state by setting $\mathsf{st}_h := \gamma_t$. We require that the transcript of the protocol is publicly decodable, so that after the T actions are performed, anybody can learn the (shared) output by inspecting st.

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

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

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

Pair-wise correlations: As observed also in [GIS18], the point of the first round OT messages was to set up pair-wise correlations between parties that were then exploited in the second round to facilitate the transfer of a bit from party j^*'s garbled circuit to party j's garbled circuit. For simplicity, assume for now that when generating the first round, the parties j and j^* already know

the bit γ_t that is to be communicated during action t. This is clearly not the case, but this issue is addressed in [GS18,GIS18] (and here) by generating four sets of correlations, corresponding to each of the four possible settings of the two bits of the public state (α, β) at the indices (f, g) corresponding to action $\phi_t = (j^*, f, g, h)$.

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

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

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

Tying loose ends: This approach works, except that, as alluded to before,

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

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

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

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

2.2 Step 2: From scHSS to FMS MPC

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

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

Garbled circuits to the rescue. Using garbled circuits, we are able to squish the above protocol to operate in only *two* rounds. The main idea is to have each party i additionally send a garbled circuit $\widetilde{\mathsf{C}}_i$ in the second round. Each $\widetilde{\mathsf{C}}_i$ garbles a circuit that implements $\mathsf{Eval}(i, \mathsf{C}, \cdot)$. Given the labels for sh_i, $\widetilde{\mathsf{C}}_i$ can be evaluated to output $y_i \leftarrow \mathsf{Eval}(i, \mathsf{C}, \mathsf{sh}_i)$. Note that, if it is ensured that every party receives all the garbled circuits *and* all the correct labels after the second round, they can obtain all $(y_1, \ldots, y_n)$, and compute the final output y without further communication. The only question left now is how the correct labels are communicated within two rounds.

Tweaking vanilla MPC to output labels. For communicating the correct labels, we slightly tweak the functionality computed by the vanilla MPC protocol. In particular, instead of using it just to compute the shares $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$, we have the vanilla MPC protocol compute a slightly different functionality that first computes the shares, and rather than outputting them directly, outputs the corresponding correct labels for everyone's shares.[7] This is enabled by having

[7] It is important to note that the set of garbled labels corresponding to some input x hides the actual string x. Hence, outputting all the labels instead of specific shares

each party provide a random value r_i, which is used to generate the labels, as an additional input to D. Therefore, everyone's correct labels are now available after the completion of the second round of the vanilla MPC protocol. Recall that parties also broadcast their garbled circuits along with the second round of the vanilla MPC. Each party i, on receiving all $\widetilde{\mathsf{C}}_1, \ldots \widetilde{\mathsf{C}}_n$ and all correct labels, evaluates to obtain $(y_1, \ldots, y_n)$ and then computes the final output y.[8]

2.3 Step 3: From FMS MPC to Reusable MPC

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

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

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

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

Intuitively, parties iteratively use an FMS instance at a particular level of the binary tree (starting from the root) to generate two new first-round FMS

---

enables everyone to obtain the desired output without any further communication, but also does not compromise security.

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

messages corresponding to the next level of the tree. The leaf FMS protocol instances will be used to compute the actual circuits. The root to leaf path traversed to compute a circuit C is decided based on the description of the circuit C itself.[9]

In more detail, parties first send the second round message of the root (0'th) level FMS protocol instance for a fixed circuit N (independent of the circuit C to be computed) that samples and outputs "left" and "right" MPC_1 messages using the same inputs that were used in the root level FMS. Now, depending on the first bit of the circuit description, parties choose either the left (if the first bit is 0) or the right (if the first bit is 1) MPC_1 messages for the next (1st) level. Now using the chosen FMS messages, parties generate the MPC_2 message for the same circuit N as above. This results in two more fresh instances of the MPC_1 messages for the next (2nd) level. As mentioned before, this procedure is continued until the leaf node is reached. At that point the MPC_2 messages are generated for the circuit C that the parties are interested in computing.

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

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

Communicating the labels for each party's garbled circuit. The trick here is (again very similar to step 2) to tweak the circuit N, in that instead of outputting the two MPC_1 messages for the next level, N (with an additional

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

random input r_i from each party i) now outputs labels corresponding to the messages.[10]

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

Adaptivity in the choice of circuit. Our reusable two-round MPC protocol satisfies a strong adaptive security guarantee. In particular, the adversary may choose any circuit to compute after seeing the first round messages (and even after seeing the second round messages for other circuits computed on the same inputs). This stronger security is achieved based on the structure of our construction, since the first round messages of the FMS MPC used to compute the actual circuit are only revealed when the actual execution happens in the second round of the reusable protocol. In particular, we do not even have to rely on "adaptive" security of the underlying FMS protocol to achieve this property.[11]

3 Preliminaries

For standard cryptographic preliminaries, see the full version [BGMM20].

3.1 Two-Round MPC

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

Definition 5 (Two-Round MPC Procotol). An n-party two-round MPC protocol is described by a triplet of PPT algorithms $(\mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$ with the following syntax.

- $\mathsf{MPC}_1(1^\lambda, \mathsf{CRS}, \mathsf{C}, i, x_i; r_i) =: (\mathsf{st}_i^{(1)}, \mathsf{msg}_i^{(1)})$: Takes as input 1^λ, a common random/reference string CRS, (the description of) a circuit C to be computed, identity of a party $i \in [n]$, input $x_i \in \{0,1\}^*$ and randomness $r_i \in \{0,1\}^\lambda$ (we drop mentioning the randomness explicitly when it is not needed). It outputs party i's first message $\mathsf{msg}_i^{(1)}$ and its private state $\mathsf{st}_i^{(1)}$.
- $\mathsf{MPC}_2(\mathsf{C}, \mathsf{st}_i^{(1)}, \{\mathsf{msg}_j^{(1)}\}_{j\in[n]}) \to (\mathsf{st}_i^{(2)}, \mathsf{msg}_i^{(2)})$: Takes as input (the description of) a circuit[12] C to be computed, the state[13] of a party $\mathsf{st}_i^{(1)}$, and the first

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

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

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

[13] Without loss of generality we may assume that the MPC_2 algorithm is deterministic given the state $\mathsf{st}_i^{(1)}$. Any randomness needed for the second round could be included

round messages of all the parties $\{\mathsf{msg}_j^{(1)}\}_{j\in[n]}$. It outputs party i's second round message $\mathsf{msg}_i^{(2)}$ and its private state $\mathsf{st}_i^{(2)}$.

– $\mathsf{MPC}_3(\mathsf{st}_i^{(2)}, \{\mathsf{msg}_j^{(2)}\}_{j\in[n]}) =: y_i$: Takes as input the state of a party $\mathsf{st}_i^{(2)}$, and the second round messages of all the parties $\{\mathsf{msg}_j^{(2)}\}_{j\in[n]}$. It outputs the ith party's output y_i.

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

The formal security definition is provided in the full version [BGMM20].

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

Definition 6 (First Message Succinct Two-Round MPC). *Let* $\pi = (\mathsf{MPC}_1, \mathsf{MPC}_2, \mathsf{MPC}_3)$ *be a two-round MPC protocol. Protocol* π *is said to be first message succinct if algorithm* MPC_1 *does not take as input the circuit* C *being computed. More specifically, it takes an input of the form* $(1^\lambda, \mathsf{CRS}, i, x_i; r_i)$.

Note that a first message succinct two-round MPC satisfies the same correctness and security properties as the (vanilla) two-round MPC.[14]

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

in $\mathsf{st}_i^{(1)}$. Even in the reusable (defined later) case, it is possible to use a PRF computed on the input circuit to provide the needed randomness for the execution of MPC_2.

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

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

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

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

- **Phase I:** All the honest parties $i \notin I$ execute the first round of the protocol by running the algorithm MPC_1 using their respective input x_i. They broadcast their first round message $\mathsf{msg}_i^{(1)}$ and preserve their secret state $\mathsf{st}_i^{(1)}$. Then the adversary $\mathcal{A}$ sends the first round messages on behalf of the corrupted parties following any arbitrary (polynomial-time computable) strategy (a semi-honest adversary follows the protocol behavior honestly and runs the algorithm $\mathsf{MPC}_1(\cdot)$).
- **Phase II (Reusable):** The adversary outputs a circuit C, which is provided to all parties.
  Next, each honest party computes the algorithm MPC_2 using this circuit C (and its secret state $\mathsf{st}_i^{(1)}$ generated as the output of MPC_1 in Phase I). Again, adversary $\mathcal{A}$ sends arbitrarily computed (in PPT) second round messages on behalf of the corrupt parties. The honest parties return the output of MPC_3 executed on their secret state and the received second round messages.
  The adversary $\mathcal{A}$ decides whether to continue the execution of a different computation. If yes, then the computation returns to the beginning of phase II. In the other case, phase II ends.

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

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

1. **Send inputs to the trusted party:** Each honest party sends its input to the trusted party. Each corrupt party P_i, (controlled by Sim) may either send its input x_i or send some other input of the same length to the trusted party. Let x_i' denote the value sent by party P_i. Note that for a semi-honest adversary, $x_i' = x_i$ always.
2. **Adversary picks circuit:** Sim sends a circuit C to the ideal functionality which is also then forwarded to the honest parties.

3. **Trusted party sends output to the adversary:** The trusted party computes $\mathsf{C}(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends $\{y_i\}_{i \in I}$ to the adversary.
4. **Adversary instructs trusted party to abort or continue:** This is formalized by having the adversary Sim send either a continue or abort message to the trusted party. (A semi-honest adversary never aborts.) In the latter case, the trusted party sends to each uncorrupted party P_i its output value y_i. In the former case, the trusted party sends the special symbol $\perp$ to each uncorrupted party.
5. **Reuse:** The adversary decides whether to continue the execution of a different computation. In the yes case, the ideal world returns to the start of Step 2.
6. **Outputs:** Sim outputs an arbitrary function of its view, and the honest parties output the values obtained from the trusted party.

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

Definition 7. *Let λ be the security parameter. Let π be an n-party two-round protocol, for $n \in \mathsf{N}$. We say that π is a reusable two-round MPC protocol in the presence of malicious (resp., semi-honest) adversaries if for every PPT real world adversary (resp., semi-honest adversary) $\mathcal{A}$ there exists a PPT ideal world adversary (resp., semi-honest adversary) Sim such that for any $\vec{x} = \{x_i\}_{i \in [n]} \in (\{0,1\}^*)^n$, any $z \in \{0,1\}^*$, any $I \subset [n]$ and any PPT distinguisher $\mathcal{D}$, we have that*

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

is negligible in λ.

4 Step 1: Constructing Sharing-Compact HSS from HSS

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

4.1 Sharing-Compact Homomorphic Secret Sharing

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

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

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

Definition 8 (Sharing-compact Homomorphic Secret Sharing (scHSS)). *A* scHSS *scheme for a class of programs $\mathcal{P}$ is a triple of PPT algorithms (*Share,Eval,Dec*) with the following syntax:*

$\mathsf{Share}(1^\lambda, n, x)$ *Takes as input a security parameter 1^λ, a number of parties n, and a secret $x \in \{0,1\}^*$, and outputs shares $(x_1, \ldots, x_n)$.*
$\mathsf{Eval}(j, P, x_j)$*: Takes as input a party index $j \in [n]$, a program P, and share x_j, and outputs a string $y_j \in \{0,1\}^*$.*
$\mathsf{Dec}(y_1, \ldots, y_n)$*: Takes as input all evaluated shares $(y_1, \ldots, y_n)$ and outputs $y \in \{0,1\}^*$.*

The algorithms satisfy the following properties.

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

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

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

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

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

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

4.2 Conforming Protocol

In our construction, we need a modification of the notion of conforming MPC protocol from [GS18]. Consider an MPC protocol Φ between parties $P_1, \ldots, P_n$. For each $i \in [n]$, we let $x_i \in \{0,1\}^m$ denote the input of party P_i. We consider any random coins used by a party to be part of its input (we can assume each party

uses at most λ bits of randomness, and expands as necessary with a PRF). A conforming protocol Φ is defined by functions inpgen, gen, post, and computation steps or what we call *actions* $\phi_1, \cdots \phi_T$. The protocol Φ proceeds in three stages: the input sharing stage, the computation stage, and the output stage. For those familiar with the notion of conforming protocol from [GS18,GIS18], we outline the differences here.

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

Next, we give our description of a conforming protocol.

- **Input sharing phase**: Each party i chooses random coins $s_i \leftarrow \{0,1\}^\lambda$, computes $(w_i, r_i) \coloneqq \mathsf{inpgen}(x_i, s_i)$ where $w_i = x_i \oplus r_i$, and broadcasts w_i. Looking ahead to the proof of Lemma 9, we will take s_i to be the seed of a $\mathsf{PRF}(s_i, \cdot) : \{0,1\}^* \to \{0,1\}$.
- **Computation phase**: Let T be a parameter that depends on the circuit C to be computed. Each party sets the global public state

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

  and generates their secret state $v_i \coloneqq \mathsf{gen}(i, s_i)$.[15] Let ℓ be the length of st or v_i (st and v_i will be of the same length). We will also use the notation that for index $f \in [\ell]$, $v_{i,f} \coloneqq \mathsf{gen}_f(i, s_i)$.
  For each $t \in \{1 \cdots T\}$ parties proceed as follows:
  1. Parse action ϕ_t as (i, f, g, h) where $i \in [n]$ and $f, g, h \in [\ell]$.
  2. Party P_i computes *one* NAND gate as

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

     and broadcasts γ_t to every other party.
  3. Every party updates st_h to the bit value γ_t received from P_i.

  We require that for all $t, t' \in [T]$ such that $t \neq t'$, we have that if $\phi_t = (\cdot, \cdot, \cdot, h)$ and $\phi_{t'} = (\cdot, \cdot, \cdot, h')$ then $h \neq h'$ (this ensures that no state bit is ever overwritten).
- **Output phase**: Denote by $\Gamma = (\gamma_1, \ldots, \gamma_T)$ the transcipt of the protocol, and output $\mathsf{post}(\Gamma)$.

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

Lemma 9. *For any input length m, there exists a function* inpgen *such that any n party MPC protocol Π (where each party has an input of length at most m) can be written as a conforming protocol $\Phi = (\mathsf{inpgen}, \mathsf{gen}, \mathsf{post}, \{\phi_t\}_{t\in T})$ while inheriting the correctness and the security of the original protocol.*

The proof of this lemma is very similar to the proof provided in [GS18] and is deferred to the full version [BGMM20].

4.3 Our Construction

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

Ingredients: We use the following ingredients in our construction.

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

Theorem 10. *Assuming a semi-honest MPC protocol (with any number of rounds) that can compute any polynomial-size functionality, a homomorphic secret sharing scheme supporting evaluations of circuits in NC^1, and a* PRF *that can be computed in NC^1, there exists a sharing-compact homomorphic secret sharing scheme supporting the evaluation of any polynomial-size circuit.*

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

(i) Circuit F. The speaking garbled circuit and the listening garbled circuit need shared secrets for communication. Using HSS, F provides an interface for setting up these shared secrets. More specifically, consider a speaking party j^*

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

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

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

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

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

Fig. 1: The Circuit F.

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

Next, we'll see how the circuit F enables communication between garbled circuits. In our construction the speaking party will just output the evaluation of F on its share (for appropriate choices of t, α, β and p). On the other hand, party j will encrypt the zero-label for its next garbled circuit using the output of the evaluation of F on its share (for appropriate choices of t, α, β and p) and will encrypt the one-label for its next garbled circuit using the exclusive or of this value and the difference value. Observe that the output of the speaking circuit will be exactly the key used to encrypt the label corresponding to the bit sent by j^* in the t^{th} action.

Finally, we need to ensure that each circuit G evaluated under the HSS can be computed in NC^1. Observe that G essentially only computes $\mathsf{gen}_f(j^*, s_{j^*})$ evaluations and $\mathsf{PRF}(k_j, \cdot)$ evaluations. The proof of Lemma 9 shows that $\mathsf{gen}_f(j^*, s_{j^*})$ may be computed with a single PRF evaluation using key s_{j^*}. Thus, if we take each s_j, k_j to be keys for a PRF computable in NC^1, it follows that G will be in NC^1.

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

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

Input: st.

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

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

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

Fig. 2: The Speaking Circuit P^*.

(iii) The Listening Circuit P. The construction of the listening circuit mirrors that of the speaking circuit. The listening circuit outputs the labels for all wires except the h^{th} wire that it is listening on. For the h^{th} wire, the listening circuit outputs encryptions of the two labels under two distinct keys, where one of them will be output by the speaking circuit during this action. As in the case of

speaking circuits, for technical reasons in the proof, we cannot have the listening circuit compute these value but must instead hard-code them. More specifically, we hard code $\left\{z_{p,0}^{(\alpha,\beta)}, z_{p,1}^{(\alpha,\beta)}\right\}_{\substack{\alpha,\beta\in\{0,1\},\\ p\in[\lambda]}}$ where $z_{p,0}^{(\alpha,\beta)}$ is obtained as $\mathsf{F}[t,\alpha,\beta,p](\mathsf{sh})$ and $z_{p,1}^{(\alpha,\beta)}$ is obtained as $z_{p,0}^{(\alpha,\beta)} \oplus \mathsf{PRF}\left(k_j,(t,\alpha,\beta,p)\right)$.

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

Input: st.

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

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

Fig. 3: The Listening Circuit P.

The Construction Itself: The foundation of a sharing-compact HSS for evaluating circuit C is a conforming protocol Φ (as described earlier in Section 4.2) computing the circuit C. Very roughly (and the details will become clear as we go along), in our construction, the Share algorithm will generate secret shares of the input x for the n parties. Additionally, the share algorithm generates the first round GS MPC messages on behalf of each party. The Eval algorithm will roughly correspond to the generation of the second round messages of the GS MPC protocol. Finally, the Dec algorithm will perform the reconstruction, which corresponds to the output computation step in GS after all the second round messages have been sent out.

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

On input x, the share algorithm generates a secret sharing of x (along with the randomness needed for the execution of Φ) to obtain a share x_j for each virtual party $j \in [n']$. In addition, two PRF keys s_j, k_j for each virtual party $j \in [n']$ are sampled. Now, the heart of the sharing algorithm is the generation of HSS shares of $\{s_j, s_{j'}\}, \{k_j, k_{j'}\}$ for every pair of $j \neq j' \in [n']$, which are then

provided to parties j and j'. Specifically, the algorithm computes shares $\mathsf{sh}_j^{\{j,j'\}}$ and $\mathsf{sh}_{j'}^{\{j,j'\}}$ as the output of $\mathsf{HSS.Share}\left(1^\lambda, (\{s_j, s_{j'}\}, \{k_j, k_{j'}\})\right)$. Note that we generate only one set of shares for each j, j' and the ordering of j and j' is irrelevant (we use the set notation to signify this).

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

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

The Evaluation Algorithm: Observe that the sharing algorithm is independent of the conforming protocol Φ (and the circuit C to be computed), thus achieving sharing compactness. This is due to the fact that the function inpgen is *universal* for conforming protocols Φ (as explained in Section 4.2).

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

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

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

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

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

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

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

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

i. Parse ϕ_t as (j^*, f, g, h).

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

$$\mathsf{arg}_1 := \left\{\mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_{j^*}^{\{j^*,j\}}\right)\right\}_{\substack{j\in[n']\setminus\{j^*\},\\ \alpha,\beta\in\{0,1\},\\ p\in[\lambda]}}$$

$$\mathsf{arg}_2 := (v_{j^*,f}, v_{j^*,g}, v_{j^*,h})$$

$$\left(\widetilde{\mathsf{P}}^{j^*,t}, \overline{\mathsf{lab}}^{j^*,t}\right) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{P}^*\left[j^*, \mathsf{arg}_1, \mathsf{arg}_2, \overline{\mathsf{lab}}^{j^*,t+1}\right]\right).$$

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

$$\mathsf{arg}_1 := \left\{\begin{array}{c}\mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_j^{\{j^*,j\}}\right),\\ \mathsf{F}[t,\alpha,\beta,p]\left(\mathsf{sh}_j^{\{j^*,j\}}\right) \oplus \mathsf{PRF}\left(k_j, (t,\alpha,\beta,p)\right)\end{array}\right\}_{\substack{\alpha,\beta\in\{0,1\},\\ p\in[\lambda]}}$$

$$\mathsf{arg}_2 := (f, g, h)$$

$$\left(\widetilde{\mathsf{P}}^{j,t}, \overline{\mathsf{lab}}^{j,t}\right) \leftarrow \mathsf{Garble}\left(1^\lambda, \mathsf{P}\left[j, \mathsf{arg}_1, \mathsf{arg}_2, \overline{\mathsf{lab}}^{j,t+1}\right]\right).$$

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

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

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

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

1. For each $i \in [n]$, parse y_i as $\left\{\left(\left\{\widetilde{\mathsf{P}}^{j,t}\right\}_{t\in[T]}, \left\{\mathsf{lab}_k^{j,1}\right\}_{k\in[\ell]}\right)\right\}_{j\in[(i-1)\lambda+1,\cdots,i\lambda]}$.
2. For each $j \in [n']$, let $\widetilde{\overline{\mathsf{lab}}}^{j,1} := \left\{\mathsf{lab}_k^{j,1}\right\}_{k\in[\ell]}$.
3. For each t from 1 to T,
   (a) Parse ϕ_t as (j^*, f, g, h).
   (b) Compute $\left(\gamma_t, \left\{z_{j,p}^*\right\}_{\substack{j\in[n']\setminus\{j^*\},\\ p\in[\lambda]}}, \widetilde{\overline{\mathsf{lab}}}^{j^*,t+1}\right) := \mathsf{GEval}\left(\widetilde{\mathsf{P}}^{j^*,t}, \widetilde{\overline{\mathsf{lab}}}^{j^*,t}\right)$.

(c) For each $j \neq j^*$:

i. Compute $\left(\{\mathsf{elab}_{p,0}, \mathsf{elab}_{p,1}\}_{p\in[\lambda]}, \left\{\mathsf{lab}_k^{j,t+1}\right\}_{k\in[\ell]\setminus\{h\}}\right) \coloneqq \mathsf{GEval}\left(\widetilde{\mathsf{P}}^{j,t}, \widetilde{\mathsf{lab}}^{j,t}\right)$.

ii. If there exists $p \in [\lambda]$, such that $\mathsf{rob.dec}\left(z^*_{j,p}, \mathsf{elab}_{p,\gamma_t}\right) \neq \perp$, then set the result to $\mathsf{lab}_h^{j,t+1}$. If all λ decryptions give $\perp$, then output $\perp$ and abort.

iii. Set $\widetilde{\mathsf{lab}}^{j,t+1} \coloneqq \left\{\mathsf{lab}_k^{j,t+1}\right\}_{k\in[\ell]}$.

4. Set $\Gamma = (\gamma_1, \ldots, \gamma_T)$ and output $\mathsf{post}(\Gamma)$.

The proof of correctness, security and robustness can be found in the full version [BGMM20].

5 Step 2: FMS MPC from Sharing-Compact HSS

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

Label Encryption. This is an encryption scheme designed specifically for encrypting a grid of $2 \times \ell$ garbled input labels corresponding to a garbled circuit with input length ℓ. The encryption algorithm takes as input a $2 \times \ell$ grid of strings (labels) along with a $2 \times \ell$ grid of keys. It encrypts each label using each corresponding key, making use of a *robust* private-key encryption scheme $(\mathsf{rob.enc}, \mathsf{rob.dec})$. It then randomly permutes each pair (column) of ciphertexts, and outputs the resulting $2 \times \ell$ grid. On the other hand, decryption only takes as input a set of ℓ keys, that presumably correspond to exactly one ciphertext per column, or, exactly one input to the garbled circuit. The decryption algorithm uses the keys to decrypt exactly one label per column, with the robustness of $(\mathsf{rob.enc}, \mathsf{rob.dec})$ ensuring that indeed only one ciphertext per column is able to be decrypted. The random permutations that occur during encryption ensure that a decryptor will recover a valid set of input labels *without knowing* which input they actually correspond to. This will be crucial in our construction.

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

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

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

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

Main Ingredients:

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

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

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

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

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

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

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

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

Circuit D

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

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

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

Fig. 5: The (randomized) circuit D

Theorem 11. *Let $\mathcal{X} \in \{$semi-honest in the plain model, semi-honest in the common random/reference string model, malicious in the common random/reference string model$\}$. Assuming a (vanilla) $\mathcal{X}$ two-round MPC protocol and a* scHSS *scheme for polynomial-size circuits, there exists an $\mathcal{X}$ first message succinct two-round MPC protocol.*

The proof of Theorem 11 can be found in the full version [BGMM20].

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

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

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

Now, it remains to show how the m (garbled circuit, encrypted labels) pairs output by each party in round two can be used to reconstruct each of the m MPC outputs, culminating in C. We use a repeated application of the mechanism developed in the last section. In particular, the d'th garbled circuit output by party i computes their second round message of the d'th MPC. The input labels are encrypted using randomness derived from party i's root randomness r_i. Specifically, as in last section, we use a PRF to compute a $2 \times \ell$ grid of keys, which will be used to LabEnc the $2 \times \ell$ grid of input labels. The key to this PRF will be generated by a PRG $(\mathsf{H}_0, \mathsf{H}_1)$ applied to $r_{i,d-1}$. Since we are branching based on the bit $p[d]$, the key will be set to $\mathsf{H}_{p[d]}(r_{i,d-1})$.

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

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

round (state, message) pair, encrypt the state with their own secret key, and then output the encrypted state plus the message in the clear. In the output phase, each party i can decrypt their own state, (but not anyone else's) and use their state to reconstruct the output of each MPC.

The formal construction and the proof of the following theorem are deferred to the full version [BGMM20].

Theorem 12. *Let $\mathcal{X} \in \{$semi-honest in the plain or CRS model, malicious in the CRS model$\}$. Assuming a first message succinct $\mathcal{X}$ two-round MPC protocol, there exists an $\mathcal{X}$ reusable two-round MPC protocol.*

7 Acknowledgements

We thank Saikrishna Badrinarayanan for valuable contributions while collaborating during the early stages of this work.

References

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

AJJ20. Prabhanjan Ananth, Abhishek Jain, and Zhengzhong Jin. Multiparty homomorphic encryption (or: On removing setup in multi-key fhe). Cryptology ePrint Archive, Report 2020/169, 2020. `https://eprint.iacr.org/2020/169`.

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

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

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

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

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

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

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

BGMM20. James Bartusek, Sanjam Garg, Daniel Masny, and Pratyay Mukherjee. Reusable two-round mpc from ddh. Cryptology ePrint Archive, Report 2020/170, 2020. `https://eprint.iacr.org/2020/170`.

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

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

BL20. Fabrice Benhamouda and Huijia Lin. Multiparty reusable non-interactive secure computation. Cryptology ePrint Archive, Report 2020/221, 2020. `http://eprint.iacr.org/2020/221`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

GS18. Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499. Springer, Heidelberg, April / May 2018.

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

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

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

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

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

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

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