

Breaking the n^2 Barrier: Information-Theoretic MPC with Sub-Quadratic Communication

Alexander Bienstock¹, Yuval Efron², and Kevin Yeo³

¹ JPMorgan AI Research & AlgoCRYPT CoE

² Columbia University

³ Google

Abstract. Secure multi-party computation (MPC) considers the problem of securely computing a function across n parties such that no adversary controlling t corrupted parties may learn any additional information beyond the function output. For information-theoretic security in the honest majority setting, recent works have made tremendous progress in reducing the MPC communication costs for large circuits obtaining as low as $\tilde{O}(|C|)$ total communication. To date, all information-theoretic MPC protocols require an additive $\tilde{\Omega}(n^2)$ communication independent of the circuit size that becomes the dominant cost for small circuits.

In this work, we consider the problem of building more efficient information-theoretic MPC protocols with $o(n^2)$ communication for small circuits. For corruption threshold $t < (1/2 - \epsilon) \cdot n$, we present an information-theoretic MPC protocol secure against adaptive and malicious adversaries with subquadratic communication for a wide range of circuits, including those in which the number of input parties I satisfies $|I| = o(n)$, circuit size $|C| = o(n^{1.5}/|I|)$, and circuit depth $\text{depth}(C) = o(n)$; for example, $|C| = o(n^{3/4})$ and $|I| = o(n^{3/4})$ input parties. To our knowledge, this is the first information-theoretic MPC protocol with $o(n^2)$ communication for a wide-range of circuit classes. Furthermore, we note the requirement of sublinear input parties $|I| = o(n)$ is necessary to obtain $o(n^2)$ communication due to the $\Omega(|I| \cdot n)$ lower bound by Damgård *et al.* [EUROCRYPT'16]. To obtain our new MPC protocol, we develop new techniques for key MPC subroutines including randomness generation, player virtualization and output reconstruction with $o(n^2)$ communication leveraging the sublinear number of input parties, $|I| = o(n)$, and randomized output distribution with unpredictable communication patterns.

Finally, we also prove a separation showing that $o(n^2)$ communication MPC is impossible for adversaries with stronger than standard adaptive rushing capabilities even for constant-size circuits.

1 Introduction

Secure Multiparty Computation (MPC) is a topic at the heart of cryptography [Yao82, GMW87, CCD88, BGW88]. MPC at a high level allows n parties to jointly compute a function of their inputs, while revealing nothing besides the output. In more detail, an adversary that corrupts t out of the n parties should

not be able to infer any information about the inputs of the remaining $n - t$ honest parties, except what may already be learned from the output. The adversary observes all the information available to the t corrupted parties, including their inputs as well as any messages sent or received from other honest parties. Adversaries in MPC are commonly categorized in several different ways. First, adversaries may corrupt in either a *static* or *adaptive* method. Static adversaries must choose the t corrupt parties before the MPC protocol commences. Adaptive adversaries are given more flexibility with the ability to corrupt parties during the protocol execution using any information observed such as messages received or sent by already corrupted parties. Additionally, adversaries may be categorized as *semi-honest* or *malicious*. For the former categorization, semi-honest adversaries must follow the protocol as specified but may use all observed information to learn information about any honest party’s input. In contrast, malicious adversaries are allowed to deviate from the protocol with the goal of learning an honest party’s inputs or forcing an honest party to output the wrong answer. Lastly, one can also consider computational or information-theoretic security where computational security considers efficient (probabilistically polynomial time) adversaries and information-theoretic security considers computationally unbounded adversaries. In this work, we will consider information-theoretic (statistical) security.

In terms of efficiency, a primary resource of interest is the communication complexity measuring the total communication between all honest parties in the MPC protocol. In its most basic form, MPC considers the function to be computed as an arithmetic circuit C consisting of addition and multiplication gates. For information-theoretically secure MPC protocols, the communication complexity is often written in terms of the number of gates in the circuit⁴, $|C|$, as well as the number of parties, n . Since the circuit representations of many functions consist of millions of gates, the dominant terms of the communication costs are typically those that grow with the circuit size $|C|$. As a result, any terms independent of $|C|$ are usually de-emphasized. Prior works (such as [DIK10, GPS22, EGP⁺23, BY25, SY25]) have obtained communication costs depending on the circuit as small as $\tilde{O}(|C|)$, but they all still require $\tilde{\Omega}(n^2)$ communication for any circuit size.

For settings with small circuits, such as those computing simpler functions, the communication arising from the additive $\tilde{\Omega}(n^2)$ term becomes the dominant cost. It turns out that this $\tilde{\Omega}(n^2)$ communication is unavoidable in certain settings. For any set I of parties with inputs and corruption threshold t , Damgård, Nielsen, Ostrovsky and Rosén [DNOR16] presented a lower bound of $\Omega(|I| \cdot t)$ for information-theoretic MPC against semi-honest and static adversaries⁵. Furthermore, this lower bound holds even for small circuits of size $|C| = O(|I|)$ such as the AND function where each of the $|I|$ input parties receives a single

⁴ In general, addition gates can be computed without communication in most approaches. Thus, $|C|$ typically refers to the number of *multiplication* gates.

⁵ We note that a lower bound of $\Omega(n \cdot t)$ was proven in [DNOR16] by assuming all n parties have an input. It is straightforward to generalize their proof to the setting when only a subset of $|I|$ parties have inputs to obtain an $\Omega(|I| \cdot t)$ lower bound.

boolean input. As most of the previous work consider the setting in which all parties have inputs $|I| = n$ and corruption thresholds of $t = \Omega(n)$, the $\tilde{\Omega}(n^2)$ communication incurred by prior MPC protocols is necessary in this setting. Therefore, to obtain any MPC protocol with $o(n^2)$ communication, it is necessary that only a sublinear number of parties have inputs, $|I| = o(n)$. We note that prior to this work, even the task of designing a protocol for the AND function with $o(n)$ input parties and $o(n^2)$ communication remained open.

Small Input Applications. The problem where only a subset of parties receive inputs is natural in many practical settings. As an example, we can consider the setting where we start with $|I|$ parties that wish to privately compute some function over their inputs. The $|I|$ parties mutually distrust each other, but wish to avoid executing expensive MPC protocols in the dishonest majority setting (that require more communication or computational assumptions). Instead, the $|I|$ parties would like to collaborate with additional parties (without inputs) who may be leveraged to help execute the MPC protocol. Furthermore, the $|I|$ parties may view these additional parties as more trustworthy as they may be financially contracted and/or legally obligated to fulfill their role honestly. As a result, the original input parties may use the additional parties (without inputs) to instead execute a more efficient MPC protocol in the honest-majority setting. This aligns with several MPC paradigms such as server-aided MPC [KMR11, HLP11, KMR12, GLO⁺21, SV25] where clients with inputs try to offload heavy computation to servers without inputs, the YOSO MPC model [GHK⁺21, BHK⁺22, CDGK22, EMP25] where there exists many (stateless) parties without inputs that help execute the MPC protocol without the need to ever send more than a single message and fluid MPC [CGG⁺21, RS22, BEP23, DDG⁺23, BEP24] where subsets of parties (without inputs) aid in executing subsets of the MPC protocol. Finally, we note this setting may also be applicable to collaborative generation of zkSNARKs where a single client ($|I| = 1$) wants to outsource a zkSNARK proof generation to a set of powerful servers (see [OB22, CLMZ23, YCW⁺25, GGJ⁺23, LZW⁺25, HMW⁺25, GGJ⁺25, FGR⁺25, LZW⁺24]).

MPC with $o(n^2)$ Communication. We start by reviewing the known MPC protocols/techniques as well as settings where one may obtain MPC protocols with $o(n^2)$ communication in a straightforward manner. We start with the setting of adversaries with static corruption that must decide on corrupted parties before the MPC protocol starts execution. If we assume sub-optimal honest-majority corruption thresholds of $t = (1/2 - \epsilon)n$, one can easily adapt the random committee sampling technique in [HIK07] to obtain $o(n^2)$ communication. At a high level, a random committee of $O(\kappa)$ parties is chosen, where κ is the security parameter. Against static adversaries, one can prove that the random committee will contain an honest majority. Thus, this committee of $O(\kappa)$ parties may effectively execute the MPC for all n parties, with a cost independent of n . The only additional communication required is between the committees and the n parties for inputs and outputs requiring $\tilde{O}(n\kappa) = o(n^2)$ communication. However, this scheme is easily broken by an adaptive adversary that observes the sampled committee before corruptions.

Additionally, we note that one could always use strong computational assumptions such as fully homomorphic encryption (FHE) to obtain MPC with very small communication [AJL⁺12, GHS12]. However, FHE is prohibitively expensive [DPSZ12, EGP⁺23] with significant barriers to being proven adaptively-secure [KTZ13, CsW19]. As a result, we focus on information-theoretic MPC without computational assumptions.

Our Problem. Given the above landscape, we focus on the setting of information-theoretic security against adaptive and malicious adversaries when the number of input parties is sublinear, $|I| = o(n)$. To date, there are currently no information-theoretic MPC protocols that achieve $o(n^2)$ communication against adaptive and malicious adversaries even for simple functionalities. This motivates our question:

Does there exist an information-theoretically secure MPC protocol with $o(n^2)$ communication complexity against adaptive and malicious adversaries?

1.1 Our Contributions

Our Construction. As the core contribution of this paper, we present the first $o(n^2)$ communication honest-majority MPC protocol with information-theoretic security against an adaptive and malicious adversary.

Theorem 1 (Informal). *For an arithmetic circuit C over a finite field $\mathbb{F}$ with inputs from $|I|$ parties, there exists an information-theoretically secure with abort MPC protocol that computes the arithmetic circuit C in the presence of a malicious adversary that adaptively corrupts $t \leq (1/2 - \epsilon)n$ parties for constant $0 < \epsilon < 1/2$. The MPC protocol requires $\tilde{O}((|I| \cdot n) + (|I| \cdot |C| \cdot \sqrt{n}) + (\text{depth}(C) \cdot n) + (\sigma \cdot n) + n^{1.5})$ total communication where σ is the number of output wires.*

Our protocol obtains sub-quadratic communication when the number of input parties is $|I| = o(n)$ and circuits of size $|C| = o(n^{1.5}/|I|)$ with sublinear $\text{depth}(C) = o(n)$ and sublinear output wires $\sigma = o(n)$. We emphasize that, prior to our work, we are unaware of any MPC protocols with sub-quadratic communication even for very small circuits with simple functionalities such as multiplying inputs from two different parties where $|I| = 2$ and $|C| = 1$. Our work obtains $o(n^2)$ communication for this problem along with a wide-range of more complex circuit classes as well.

Furthermore, we note that the requirement that $|I| = o(n)$ is necessary for $o(n^2)$ communication. In particular, a lower bound of $\Omega(|I| \cdot t)$ was shown in [DNOR16] even for circuits of size $|C| = O(|I|)$ and $\text{depth}(C) = \tilde{O}(1)$. Since we consider corruption thresholds of $t \leq (1/2 - \epsilon) \cdot n$, any MPC protocol must have communication $\Omega(|I| \cdot n)$ matching the first term in our protocol's cost.

Lower Bounds. We also present a lower bound showing a separation of sub-quadratic communication for stronger adversaries. In our above MPC construction, we consider standard, malicious rushing adversaries in UC security. One can consider stronger rushing adversaries that may observe messages from honest parties in a communication round, adaptively corrupt them in the middle of

that round, and then change all of their messages. In contrast, standard rushing adversaries cannot change any messages sent by adaptively corrupted honest parties after observing their original messages for that round (the adversary may only choose the messages from already corrupted parties with this information). We show $o(n^2)$ communication is impossible for these stronger adversaries:

Theorem 2 (Informal). *Consider any MPC protocol that is secure with abort against adversaries with adaptive, strong rushing corruptions that corrupt up to $t = \Omega(n)$ parties. Then the MPC protocol requires $\Omega(n^2)$ communication.*

In particular, our lower bound holds even for circuits of size $|C| = 1$ with just a single multiplication gate. We also show that our lower bound extends to MPC protocols with deterministic (or predictable) communication patterns even against standard malicious adversaries that explains the required $\Omega(n^2)$ additive communication cost in state-of-the-art prior works [DIK10, GPS22, EGP⁺23, BY25, SY25]:

Theorem 3 (Informal). *Consider any MPC protocol with a deterministic (or predictable) communication pattern that is secure with abort against adversaries with standard adaptive rushing corruptions that corrupt up to $t = \Omega(n)$ parties. Then the MPC protocol requires $\Omega(n^2)$ communication.*

1.2 Discussion of Results

Circuits with $o(n^2)$ Communication. Our current MPC protocol’s communication cost depending on the circuit is $\tilde{O}(|I| \cdot |C| \cdot \sqrt{n}) + (\text{depth}(C) \cdot n) + (\sigma \cdot n)$. This enables $o(n^2)$ communication for a large class of circuits with size $|C| = o(n^{1.5}/|I|)$, depth $\text{depth}(C) = o(n)$ and output wires $\sigma = o(n)$. For example, our MPC protocol obtains $o(n^2)$ communication for circuits of size $|C| = o(n^{0.75})$ with inputs from $|I| = o(n^{0.75})$ parties where the number of outputs to each party is sublinear. Other examples can include larger circuits that receive inputs from a small number of parties such as circuits of size $|C| = o(n^{1.5})$ that obtain inputs from a constant number of parties $|I| = O(1)$ as long as the multiplicative depth of C and the number of outputs to all parties is sublinear.

Number of Output Parties. We note that our MPC protocol only restricts the number of input parties. Throughout our paper, we assume that every party will receive all $\sigma = o(n)$ final outputs. In certain settings, it may be required that only some of the parties obtain the outputs while the remaining parties do not learn the output. Our protocol may be easily modified such that only a predetermined set of parties can learn the outputs. For σ outputs, the cost is $O(\sqrt{n}\sigma)$ amortized communication per output party. In other words, for settings where a smaller number of parties learn the output, our protocol uses less communication while also being more applicable to related models including server-aided MPC, YOSO MPC and fluid MPC. We point readers to Section 8 for further discussion.

Corruption Threshold. In our work, we consider sub-optimal honest-majority corruption thresholds of $t = (1/2 - \epsilon) \cdot n$ for constant $0 < \epsilon < 1/2$. One could

consider MPC against more corruptions such as strict honest majority with $t = n/2 - 1$. We note our MPC protocol’s communication is smaller than possible in the strict majority setting in certain regimes. It was shown that $\Omega(|C| \cdot n)$ communication is required when $t = n/2 - 1$ even for small-depth circuits and a constant number of outputs per party [DLN19]. In contrast, our protocol uses $o(|C| \cdot n)$ surpassing the above lower bound for similar circuits of low-depth and constant number of output wires when the number of input parties is $|I| = o(\sqrt{n})$. We leave it as future work to obtain sub-quadratic communication for $t = n/2 - 1$ that will require different techniques and larger communication than our construction.

Output Guarantees. Our MPC protocol considers security with abort as opposed to the stronger guaranteed output delivery (GOD). We note that MPC with GOD and $t \geq n/3$ requires either the assumption of ideal broadcast and/or computational assumptions [LSP19], making the problem different and less clean (e.g., how much should an ideal broadcast call cost?). In contrast, we consider information-theoretic MPC without these assumptions in this work. We leave it as an open problem to obtain $o(n^2)$ MPC with GOD using stronger assumptions.

Additive $\tilde{O}(n^{1.5})$ Communication. Our construction has an additive $\tilde{O}(n^{1.5})$ communication term independent of the circuit size. This matches the cost of the state-of-the-art Byzantine Agreement (BA) protocol without additional computational assumptions known by different techniques [KS11]. To our knowledge, this is the best such BA protocol in terms of communication cost even considering weaker output guarantees akin to abort in MPC. Therefore, this can be viewed as a barrier as improving our MPC protocol could conceivably lead to an $o(n^{1.5})$ communication BA protocol, which has been an open problem for decades.

Input Party Corruption and Secure Erasures. As we study MPC in generality, we note it is possible for an adversary to corrupt all the input parties prior to executing the MPC protocol, which is unavoidable. However, as discussed earlier, there are several previously studied settings where an adversary is assumed to be unable to corrupt all input parties such as the client-server settings of server-aided MPC [KMR11, HLP11, KMR12, GLO⁺21, SV25], fluid MPC [CGG⁺21, RS22, BEP23, DDG⁺23, BEP24] and server-outsourced zkSNARK proof generation [OB22, CLMZ23, YCW⁺25, GGJ⁺23, LZW⁺25, HMW⁺25, GGJ⁺25, FGR⁺25, LZW⁺24] where some subset of clients are assumed to remain honest and uncompromised. In this case, we guarantee both privacy and security with abort for all honest parties.

We also consider a more nuanced setting where the adversary may adaptively corrupt an input party after they have committed to their input. In this case, we want to guarantee that the adversary can no longer change the party’s input. To obtain this stronger security guarantee, we assume that honest parties can securely erase components of their state once they are unneeded (as used in prior works [BDOZ11, DPSZ12, NNOB12, GIOZ17, BH93, IPS09, Lin09, HLP15]). Our protocol guarantees that an adversary cannot change inputs after parties have committed to them in the MPC protocol and securely erased certain critical parts of their state. This includes even if the adversary adaptively corrupts the

input party afterwards. We note secure erasures are only needed for this stronger guarantee. For the setting where the adversary never corrupts all input parties, then secure erasures are unnecessary. We point readers to Section 4 for more details.

2 Technical Overview

We present a high-level overview of our protocol and the ideas for our $\Omega(n^2)$ lower bound for strong adaptive rushing adversaries.

Before presenting our protocol, we deep dive into some necessary requirements of an $o(n^2)$ MPC protocol. We show that the following two facts must be true:

1. Any $o(n^2)$ communication MPC protocol must critically leverage that there are a sublinear number of input parties $|I| = o(n)$.
2. In any $o(n^2)$ communication MPC protocol, the communication pattern (consisting of the size of messages between any two parties in each round) protocol must be randomized and unpredictable for any adaptive adversary.

The first requirement follows from the known lower bound showing that any MPC protocol for circuits with inputs from $|I|$ parties must have communication $\Omega(|I| \cdot n)$ [DNOR16]. In other words, our MPC protocol must somehow leverage that the number of input parties is sublinear, $|I| = o(n)$, to obtain $o(n^2)$ communication.

To understand the second requirement, we begin by noting that the standard MPC adversary observes the entire communication pattern of the MPC protocol including the size (but not the contents) of the communication between every pair of parties in each communication round. As we consider $o(n^2)$ communication, we immediately know that it is impossible for each pair of parties to communicate with each other (which is a common pattern in MPC protocols, particularly for randomness generation and output reconstruction). As a result, there exists at least some party P that communicates with $o(n)$ other parties during the MPC protocol. If the adversary successfully identifies this party P and corrupts all $o(n) < t$ parties that send messages to party P , we note the adversary can successfully ensure that the MPC protocol cannot obtain security with abort guarantees. In particular, as the adversary controls all messages to party P , the adversary can force the party P to return a output different from all other honest parties. Therefore, an $o(n^2)$ communication MPC protocol must successfully protect against such an adversarial attack.

Looking back at state-of-the-art prior works such as [DIK10, GPS22, EGP⁺23, BY25, SY25], we note that their communication patterns are deterministic. If the communication pattern of an MPC protocol is deterministic, it is impossible to achieve security with abort with $o(n^2)$ communication as the adversary can identify a party P that receives messages from at most $o(n)$ parties, corrupt all parties sending messages to P and employ the adversarial attack described above. With deterministic communication patterns, it must be that each of the n parties receives messages from at least $t + 1$ other parties. This immediately shows that

the $(t + 1)n = \Omega(n^2)$ additive term in the communication cost of these works is necessary to obtain even security with abort (see Appendix 10 for more details).

A key takeaway from this observation is that the communication pattern of any $o(n^2)$ MPC protocol must be randomized. Even more importantly, the communication pattern must be unpredictable to the adversary that may adaptively corrupt parties through the protocol's execution. As a result, one of the novelties necessary in our $o(n^2)$ communication MPC protocol is a method to ensure an unpredictable communication pattern such that no adaptive adversary can identify all $o(n)$ parties that will communicate with any single party P .

Protocol Summary. We start by presenting a summary of our protocol:

1. *Computation using Committees:* First, our protocol enables the n parties to privately compute secret shares of the correct output using player virtualization amongst randomly-chosen committees. This critically relies on the first requirement above that the number of input parties is sublinear, $|I| = o(n)$, so that input parties can generate random coins and random sharings for the rest of the parties (note if all input parties are corrupted from the beginning, then there are no privacy or correctness guarantees). The purpose of executing the MPC in this fashion with committees is to enable a randomized communication pattern for the output phase of the protocol.
2. *Random Output Distribution:* The next part distributes the shared outputs to all n parties in a randomized manner ensuring that each of the parties can recover the correct answer or abort. In particular, this adheres to the second requirement above, since the distribution is such that an adversary cannot predict the subset of parties that is sending messages to any single party.

Computation using Committees. At a high level, the first part of our MPC protocol will consider m randomly-chosen committees each of η parties. We will use player virtualization such that each of the m committees will emulate a virtual party in a larger MPC protocol executed over the m virtual parties. Roughly speaking, our goal is to completely eliminate communication between all pairs of parties. First, we permit quadratic $O(\eta^2)$ communication within each of the m committees enabling each committee to emulate a single virtual player resulting in $O(m \cdot \eta^2)$ cost. We will show that we can construct committees with appropriate parameters of $m = \tilde{O}(\sqrt{n})$ and $\eta = \tilde{O}(\sqrt{n})$ such that this cost is $\tilde{O}(n^{1.5})$. Note, we omit poly-logarithmic factors in n for convenience using $\tilde{O}$ notation. For the larger MPC protocol executed over the m virtual parties, we will use the existing Sharing Transformations protocol [GPS22]. However, Sharing Transformation requires quadratic communication between all pairs of virtual players. If we tried a trivial approach of player virtualization, communication between any pair of virtual parties requires communication between all pairs of parties within each of the corresponding committees meaning $\tilde{O}(\eta^2)$. Across all possible $O(m^2)$ pairs of virtual parties, we would end up back at quadratic communication since $\tilde{O}(m^2 \cdot \eta^2) = \tilde{O}(n^2)$. Instead, we show that the most costly subroutines of Sharing Transformations involving random coin and sharing generation that require quadratic communication between all pairs of parties

may be replaced with new protocols such that each of the m virtual parties only needs to communicate with the $|I|$ input parties. As a result, the communication cost becomes $\tilde{O}(m \cdot \eta \cdot |I|) = \tilde{O}(n \cdot |I|) = o(n^2)$ where we critically rely on the fact that $|I| = o(n)$. We elaborate on each component of this blueprint below that enables us to construct an $o(n^2)$ communication MPC protocol.

We start with our construction of m committees consisting of η parties. To enable Sharing Transformations with small enough communication for our usage, we require that at most $\delta \cdot m$ of the virtual parties (committees) are dishonest, where $\delta > 0$ is an arbitrarily small constant. We say a committee is honest if at most $(1/2 - \epsilon') \cdot \eta$ parties in the committee are corrupted by the adversary, where $0 < \epsilon' < \epsilon$ is an arbitrary constant. To construct our committees, we extend the ideas of using samplers that have been previously used to obtain sub-quadratic communication Byzantine agreement by King and Saia [KS11]. At a high level, a sampler function is $H : [m] \rightarrow [n]^\eta$ where $H(x)$ is a set of η parties from $[n]$ representing the x -th committee for all $x \in [m]$. King and Saia [KS11] used a non-explicit sampler construction guaranteeing that strictly more than half the committees will be honest for every choice of corrupted parties. We improve upon this idea by presenting an explicit randomized construction of a sampler with $m = \tilde{O}(\sqrt{n})$ committees consisting of $\eta = \tilde{O}(\sqrt{n})$ parties each that satisfies the required properties except with negligible probability (see Section 6).

Before even considering the Sharing Transformations protocol, we note that the above committee sampling approach already requires shared randomness generation amongst all n parties. It turns out that all known adaptively secure coin flipping protocols already require $\Omega(n^2)$ communication (see [FT25] for example). As a result, it is unclear whether it is even possible to sample our committees in $o(n^2)$ communication. Here, we highlight our new technique that essentially leverages the sublinear number of input parties. In particular, we rely upon the $|I| = o(n)$ input parties as *leaders* that are responsible for each generating and distributing shares of random values to all n parties. As long as at least one input party is honest, we show that the resulting shared randomness is uniformly random. However, one may quickly note that the adversary is able to corrupt $t = \Omega(n)$ parties and there are only $|I| = o(n)$ input parties. Therefore, an adversary can simply corrupt all input parties and guarantee that the shared randomness generation fails. However, we rely upon the critical observation that there are no longer any privacy or correctness requirements once all the input parties are compromised (beyond the fact that all honest parties output the same value, or abort). Indeed, the adversary has already observed all private inputs to the protocol. Furthermore, the adversary is free to change the inputs to guarantee that the honest parties receive any output of the adversary's choosing. Therefore, the only case where we must consider security is when at least one input party is honest that corresponds to when our shared randomness generation is successful. As a note, it turns out that our committee assignments require a very large amount of information that would already result in quadratic communication, if all input parties had to send these assignments to the other parties. Instead, our techniques enable one input party, say P_1 , to send the pertinent committee

information to each party, while the other input parties use a universal hash function to generate a succinct hash of this information for the parties to check P_1 's assignments, reducing the communication to $o(n^2)$.

With the above shared randomness generation, our MPC protocol can now sample committees using $o(n^2)$ communication. Additionally, we will have the $|I| = o(n)$ input parties share their inputs with all n parties requiring $\tilde{O}(|I| \cdot n) = o(n^2)$ communication. All that remains is to show that we can execute the Sharing Transformation protocol [GPS22] in $o(n^2)$ communication. It turns out that the only component of Sharing Transformations using $\Omega(n^2)$ communication is shared random coins and sharings generation (this also happens to be true for most prior MPC protocols with quadratic communication). Therefore, replacing all shared randomness generation in Sharing Transformations with our approach using the $|I| = o(n)$ input parties as leaders, we can execute Sharing Transformations with $o(n^2)$ communication if we consider very small circuits. However, we can further utilize the above idea of using the input parties as leaders to optimize communication costs for larger circuit sizes. In the end, the execution of the remaining portions of Sharing Transformation with our improvements uses $\tilde{O}(|I| \cdot |C| \cdot \sqrt{n}) + (\text{depth}(C) \cdot n)$ communication. After the execution of Sharing Transformations, each of the m virtual parties have output shares. The η parties within each honest committee may reconstruct the committee's output share to each other with $\tilde{O}(\eta^2)$ communication using error-detection. Altogether, this final committee output reconstruction requires $\tilde{O}(m \cdot \eta^2) = \tilde{O}(n^{1.5})$ communication.

Random Output Distribution. At the end of the first part of our MPC protocol, we have that each party within each of the committees holds the same corresponding virtual party output share resulting from executing Sharing Transformations over the m committees as virtual parties. We use this fact in the next part to ensure that its communication pattern is randomized and unpredictable to an adaptive adversary. As a reminder, we must ensure that no adversary may corrupt all parties sending messages to any single party P .

Our random output distribution protocol works as follows. Each party picks ℓ other parties uniformly at random to distribute their share of the output. For sufficiently large $\ell = \tilde{O}(n/\eta)$, we show that each of the n parties will receive an output share from a party within each of the $(1 - \delta)m$ honest committees except with negligible probability (see Section 8). This would allow each of the n parties to successfully reconstruct the output ignoring corrupted parties that may send incorrect output shares. This requires a total of $\tilde{O}(\ell \cdot m \cdot \eta) = \tilde{O}(n \cdot m) = \tilde{O}(n^{1.5})$ communication. As a reminder, the first step of the protocol required $\tilde{O}(m \cdot \eta^2)$ communication to reconstruct outputs after the execution of Sharing Transformations. Here, we observe that our choice of $m = \tilde{O}(\sqrt{n})$ and $\eta = \tilde{O}(\sqrt{n})$ minimizes the additive term of $\tilde{O}(n^{1.5})$ communication in our protocol.

So far, we have ignored the existence of corrupted parties that may create incorrect output shares. If any party receives multiple output shares from the same committee that are inconsistent, the party will abort. A party can also abort if the shares from the m committees do not define a valid secret sharing using error-detection techniques of standard secret sharing schemes (since δ is

arbitrarily small). As each party will receive at least one correct output share from an honest party in every honest committee, every party either reconstructs the correct output or aborts satisfying security with abort.

Finally, we critically highlight that the communication pattern of our output distribution is unpredictable to the adversary. Our random output distribution protocol works in a single communication round where every honest party picks ℓ uniformly random parties to send their output shares. Furthermore, the choice of the ℓ parties is generated using local randomness independent of all prior potential adversarial knowledge. Therefore, it is impossible for an adversary to identify the subset of honest (not currently corrupted) parties that will send their output shares to any party P better than random guess. As a reminder, in our model, we consider standard adaptive adversaries used in UC security that may observe the messages from honest parties in a communication round before choosing the messages from corrupted parties. However, by the time the adversary observes any information about the communication pattern in the output distribution, the adversary no longer has the ability to corrupt new parties. Effectively, the adversary cannot predict the subset of honest parties that will communicate with any single party P beyond random guessing.

Remaining Challenges. Finally, the above approach still comes with some additional challenges. Our key observation is that, if all input parties are corrupted, then security with abort trivially holds. While true in spirit, there are two issues that must be considered. The security notion for MPC not only requires privacy of inputs of honest parties, but also requires that all honest parties output the correct output or abort if they detect that corrupt parties do not follow the protocol. First, if all input parties are corrupted from the beginning, then the output of the protocol is completely determined by the adversary. However, if the honest parties output anything, then they must all output the same value. Second, if some input parties start honest but are adaptively corrupted later in the protocol, then the output of the protocol must be defined with respect to the original inputs of input parties (even those that were corrupted from the start). We solve the first issue by relying on a weak and efficient consolidation protocol. Each of the n parties samples $O(\sqrt{n})$ other random parties and sends them their outputs. An honest party will abort if any of the outputs they receive are different from their computed output. For the second challenge, we require input parties to sample random challenge sharings (using the random sharings generation subroutine from above) before they share their inputs to other parties.

Lower Bound Overview. We note that one could also consider more powerful adversaries with stronger adaptive rushing corruptions that may corrupt honest parties after observing their messages in a communication round, and then change any of their messages in that communication round. With respect to our random output distribution, the adversary may observe all parties sending messages to any single party P and corrupt them to change the final output of party P . We show that this barrier is inherent with these adversaries with stronger adaptive rushing corruption. We prove that any MPC protocol secure against such adversaries requires $\Omega(n^2)$ communication. At a high level, our proof techniques

goes as follows. Consider any protocol with $o(n^2)$ communication. Pick a party P uniformly at random. With non-negligible probability, party P will receive messages from at most $o(n) < t$ other parties. Within each communication round, the adversary adaptively corrupts any honest party that proposes to send a message to P using its strongly adaptive corruption capabilities. As the adversary controls all messages sent to the party P , it can compromise security with abort.

3 Related Works

Honest-Majority MPC with $t < (1/2 - \epsilon)n$. For information-theoretic MPC with sub-optimal honest-majority threshold with $t < (1/2 - \epsilon) \cdot n$, there has been a long line of work achieving small communication such as the following works and references therein [FY92, DIK10, GPS21, GIP15, GIOZ17, BGJK21, GSY21]. The recent works of [GPS21, SY25] obtain optimal $\tilde{O}(|C|)$ communication matching known lower bounds presented in [DLN19].

Committee-Based MPC. The idea of using committees of multiple parties was first introduced by Bracha [Bra87] for usage in player virtualization. Similar ideas of using committees for MPC have been used in many prior works including static adversaries [HIK07], static adversaries with computational assumptions [BBK⁺24], adaptive adversaries [NR17, BY25, SY25], optimizing garbled circuits [BEO17], optimizing GMW-style protocols [HOSS18b, HOSS18a] and reducing total computation [DIK⁺08] to list some examples.

MPC with Small Input Parties. In the past, there have been various forms of MPC studied that also consider the setting where not all parties will receive inputs, $|I| < n$. The first such notion is server-aided MPC that considers the setting where there exist clients with inputs who collaborate with one or more helper servers that are willing to securely compute functions over their inputs. Over the past decade, there has been a long line of work improving server-aided MPC (see [KMR11, HLP11, KMR12, GLO⁺21, SV25] and references therein).

A more recently studied MPC variant is the YOSO (You Only Speak Once) model introduced by Gentry *et al.* [GHK⁺21]. In the YOSO model, there exist many stateless parties (that do not receive any inputs) that are available to help with the execution of the MPC protocol. Furthermore, there is an additional restriction that these stateless parties may only be relied upon to send a single message throughout the entire protocol. In recent years, there have been several follow-up works studying YOSO MPC including [BHK⁺22, CDGK22, EMP25]. The slightly less restrictive model of fluid MPC was also introduced recently by Choudhuri *et al.* [CGG⁺21] where there exists several helper parties that will help with executing subsets of the MPC protocol. However, the set of participating parties is dynamic and each party may enter or leave the execution between rounds. We note that there have been several works studying fluid MPC such as [RS22, BEP23, DDG⁺23, BEP24].

Atomic Secure Multiplication. Cramer *et al.* [CDdH07] study the communication complexity of performing a single secure multiplication of two secret-shared elements over a finite field L in one round with $O(n)$ communication by assuming

that L contains a subfield K over which it has an extension degree linear in n . However, there are limitations including only assuming a semi-honest adversary and no full modeling of correctness for a single multiplication.

Communication Lower Bounds. Note that the usage of FHE enables MPC with even $\tilde{O}(|I| + n)$ communication [AJL⁺12, GHS12], but FHE is prohibitively expensive [DPSZ12, EGP⁺23] with substantial challenges to be proven adaptively secure [KTZ13, CSW23]. Therefore, the majority of prior communication lower bounds focus on MPC with information-theoretic security. A lower bound of $\Omega(t \cdot n)$ was presented in [DNOR16] that may be easily generalized to $\Omega(t \cdot |I|)$ for $|I| < n$ input parties. For larger circuit sizes, a lower bound of $\Omega(|C| \cdot n)$ was presented in [DLN19] with $t = n/2 - 1$ and a lower bound of $\Omega(|C|)$ when $t = (1/2 - \epsilon)n$ for any constant $0 < \epsilon < 1/2$ extending prior work considering restricted protocols [DPP14, DNPR16]. Recent work presented stronger lower bounds in the perfect security setting [DLS21].

MPC with Randomized Communication Patterns. One area of cryptography requiring randomized communication patterns is topology-hiding computation. In this setting, there exists some underlying graph that defines which pairs of parties may communicate. The goal is to securely compute functions without revealing neither the inputs nor the underlying communication graph. This was first introduced by Moran, Orlov and Richelson [MOR15]. Over the past few years, there has been a wide range of follow-up works including [HMTZ16, AM17, ALM17, BBKM23] and references therein. We note topology-hiding is orthogonal to our setting where it is known that all parties may communicate with each other, but we wish to obfuscate the actual communication pattern. Note that YOSO [GHK⁺21] and some of its followups indeed have randomized communication patterns, particularly, when participants for each round are chosen randomly. However, these models are orthogonal as they selecting participants randomly with computational assumptions and consider only random corruptions (instead of fully adaptive corruptions).

4 Preliminaries

MPC Preliminaries. We will consider multi-party computation (MPC) consisting of n parties denoted $P_1, \dots, P_n$ whose goal is to compute a functionality $\mathcal{F}$. We assume $\mathcal{F}$ is represented using an arithmetic circuit C over a finite field $\mathbb{F}$ that consists of two types of gates: addition and multiplication. Whenever we refer to the size of the arithmetic circuit $|C|$, we mean the number of multiplication gates in C . Furthermore, we will consider arithmetic circuits that receive inputs from a subset of parties denoted by $I \subseteq [n]$. We note that the subset of input parties is a part of the description of the functionality $\mathcal{F}$ as well as the arithmetic circuit C . Thus, the identity of the input parties I is fixed ahead of the protocol and known to all parties. Throughout this work, we will suppose the finite field size satisfies $|\mathbb{F}| \geq 2^\kappa$ where κ is the statistical security parameter. As we will assume that the number of parties n and circuit size $|C|$ are bounded by $\text{poly}(\kappa)$, this means the finite field size satisfies both the following inequalities: $|\mathbb{F}| \gg |C|$ and $|\mathbb{F}| \gg n$.

Next, we describe the adversarial model. We work in the Universal-Composability (UC) framework of Canetti [Can01] (see Appendix A for more details). We consider MPC that is secure against malicious and statistical adversaries that can adaptively corrupt up to $t = (1/2 - \epsilon)n$ parties for some constant $0 < \epsilon < 1/2$. In other words, the computationally unbounded adversary may choose to corrupt parties using prior information observed during the protocol and the corrupt parties are allowed to perform actions that deviate from the protocol's prescribed algorithms. We note that the adversary observes all information available to corrupt parties. Denote the set of corrupted parties at any given time as $\mathcal{T}$ and the set of honest parties as $\mathcal{H} = [n] \setminus \mathcal{T}$. Additionally, the adversary can see the communication pattern between all parties. In particular, the adversary can observe the size of communication between two honest parties in any round (including the absence of such communication). Finally, our work will consider rushing adversaries where the adversary may observe the communication from honest parties in each round before choosing the messages of corrupt parties. We note that the adversary is not allowed to corrupt honest parties to remove or change honest party messages after observing their messages (see Appendix 10 for further details and $\Omega(n^2)$ communication lower bounds for this powerful adversary even for simple functionalities). Formally, we consider the following:

Functionality 1: $\mathcal{F}_{\text{MPC}}$

Parameters: This functionality is parametrized by a number of parties n , a subset I of input parties, a corruption threshold t , and some functionality f defined over $|I|$ input values.

Corruptions: At any time, the adversary $\mathcal{A}$ can notify $\mathcal{F}_{\text{MPC}}$ that some party P has been corrupted. If at any time, the number of corruptions $|\mathcal{T}| > t$, then $\mathcal{F}_{\text{MPC}}$ outputs **abort** to all parties.

1. If the input parties are all corrupted, $I \subseteq \mathcal{T}$, before they provide $\mathcal{F}_{\text{MPC}}$ their inputs, then $\mathcal{F}_{\text{MPC}}$ lets $\mathcal{A}$ decide the output y of the functionality and a set of honest parties $\mathcal{H}_{\text{abort}} \subseteq \mathcal{H}$. $\mathcal{F}_{\text{MPC}}$ then sends **abort** to all parties in $\mathcal{H}_{\text{abort}}$ and the *same* output y to all other parties $\mathcal{H} \setminus \mathcal{H}_{\text{abort}}$.
2. Else, $\mathcal{F}_{\text{MPC}}$ collects from $\mathcal{A}$ the corrupt parties' inputs $\{x_i \mid i \in \mathcal{T} \cap I\}$. $\mathcal{F}_{\text{MPC}}$ also collects from the honest parties their inputs $\{x_i \mid i \in I \cap \mathcal{H}\}$.
3. $\mathcal{F}_{\text{MPC}}$ computes $y \leftarrow f((x_i \mid i \in I))$, and sends the result first to $\mathcal{A}$, who replies with set of honest parties $\mathcal{H}_{\text{abort}} \subseteq \mathcal{H}$. $\mathcal{F}_{\text{MPC}}$ sends **abort** to parties in $\mathcal{H}_{\text{abort}}$ and the *same* output y to parties in $\mathcal{H} \setminus \mathcal{H}_{\text{abort}}$.

Output authenticity. We note that we consider the setting where the adaptive adversary may corrupt more parties than the total number of input parties. As we study MPC in full generality, it is possible for the adversary to corrupt all input parties (even before the MPC protocol starts). In this case, one cannot provide any guarantees on the output. However, we note that there are several previously studied problem settings with more restricted adversarial corruptions

where not all input parties may be corrupted where it is critical to ensure that all honest parties output the same value. Some examples include the client-server settings of server-aided MPC [KMR11, HLP11, KMR12, GLO⁺21, SV25] and fluid MPC [CGG⁺21, RS22, BEP23, DDG⁺23, BEP24] that typically assume the adversary cannot corrupt all the clients (acting as the input parties in our notation). In these settings, our MPC protocol provides standard output guarantees for all honest parties.

Nevertheless, the above fact leads to an important but subtle component of our modeling of $\mathcal{F}_{\text{MPC}}$ is our handling of corruptions of the input parties in set I as we consider general adaptive corruption. We refer to this as the *output authenticity* problem, in which we need to determine what guarantees the outputs of honest parties should satisfy. The method for handling the corruptions of all parties in I depends on when these corruptions (adaptively) occur. What if indeed the adversary corrupts all input parties at the very beginning of the protocol? In this case, the adversary can determine all inputs, and thus the output of the functionality. We note an adversary may be able to pick an output that is outside the range of the computed function. However, it is well known that most MPC protocols suffer from additive attacks [GIP⁺14, GIP15, CGH⁺18] where adversaries may shift outputs arbitrarily (including outside of the range) if the honest parties are unable to authenticate the outputs (as is the case if all honest parties are corrupted at the start). Furthermore, there are no privacy requirements to satisfy. Yet, standard MPC output guarantees, $\mathcal{F}_{\text{MPC}}$, require that if the honest parties do not abort in this case, then they must output *the same* value.

On the other hand, we can also consider the setting where input parties start out as honest. Ideally, after the input parties have committed to their inputs to some certain extent in the protocol, the input should not be modifiable by an adversary that adaptively corrupts the input parties afterwards. Formally, our goal is to guarantee that $\mathcal{F}_{\text{MPC}}$ provides all parties with the output of f based on the inputs received from honest parties during the input phase (assuming the adversary chooses to not abort). To obtain this guarantee, we will assume secure erasures (an assumption used in prior works such as [BDOZ11, DPSZ12, NNOB12, GIOZ17, BH93, IPS09, Lin09, HLP15]) in our protocol where honest parties will delete a certain component of their private state. After this point, our protocol guarantees that an adaptive adversary can no longer change any party’s inputs even if the adversary chooses to corrupt parties afterwards. We will commonly refer to this part of the protocol as the “input phase” that we will formally define in our functionality for UC security.

This additional security requirement imposes a significant challenge on any purported protocol aiming to securely realize the functionality. The honest parties must detect any attempts by the adversary to give them inconsistent outputs, even if all input parties were corrupted at the start. If this is not the case, the honest parties must also detect any attempt by the adversary to modify the input of any party (even corrupted). We show how our MPC protocol obtains our desired *output authenticity* in Section 8. As a final note, our requirement of

secure erasure is only necessary to provide stronger security guarantees when the adversary corrupts all input parties. If at least one input party remains honest, our protocol provides security with abort without needing secure erasure.

Committees. We rely on a version of the *party virtualization* technique of Bracha [Bra87]. We will randomly sample each of the $m = \Theta(\sqrt{n})$ committees of $\eta = \Theta(\sqrt{n})$ parties (with replacement). Every committee will represent a party of a *virtual* MPC protocol, while the state of this party will be secret shared by the committee members. We will show in Section 6 that since the corruption threshold $t < (1/2 - \epsilon)n$, our random committee sampling will guarantee with all-but-negligible probability that at most $\tau < \delta m$ committees, for $\delta > 0$ an arbitrary constant, will have at least $(1/2 - \epsilon')\eta$ corrupted members, where $0 < \epsilon' < \epsilon$ is an arbitrary constant. Indeed, all other committees will have number of corrupted member parties at most $\tau' < (1/2 - \epsilon')\eta$. We will therefore call a committee *honest* if indeed the number of corrupted parties is at most τ' ; otherwise we call it *corrupted*. We denote *honest* committees as $\mathcal{H}_{\text{virt}}$ and *corrupted* committees as $\mathcal{T}_{\text{virt}}$. For a given committee $\mathcal{C}_i$, we will denote the set of honest parties in that committee as $\mathcal{H}_i$ and the set of corrupted parties in that committee as $\mathcal{T}_i$.

Secret Sharing. We use degree- d Shamir secret sharing [Sha79] over the field $\mathbb{F}$ throughout this paper, denoted by $[x]_d := (x^1, \dots, x^n)$. Shamir secret sharing gives the property that if at most d shares are held by the adversary, then the underlying secret $x \in \mathbb{F}$ is unknown to the adversary, while if any party holds at least $d + 1$ shares, they can reconstruct x (and in fact the whole sharing). Let $\alpha_1, \dots, \alpha_n$ be n distinct elements in $\mathbb{F}$. A Shamir secret sharing is constructed via a polynomial $F(y) \in \mathbb{F}[y]$ of degree at most d such that $F(0) = x$ and $F(\alpha_i) = x^i$, the i -th party's share, for all $i \in [n]$. We will also use degree- d packed secret sharing [FY92], denoted by $[\mathbf{x}]_d := (\mathbf{x}^1, \dots, \mathbf{x}^n)$. This is used to share a vector of secrets $\mathbf{x} \in \mathbb{F}^\rho$, where $\rho > 0$ is called the *packing parameter*. In this case, if at most $d - \rho + 1$ shares are held by the adversary, then $\mathbf{x}$ is unknown to them, while still, at least $d + 1$ shares are required to reconstruct $\mathbf{x}$ (and also the whole sharing). Let $\text{pos} := (p_1, \dots, p_\rho)$ be another ρ elements in $\mathbb{F}$ that are distinct from one another and $\alpha_1, \dots, \alpha_n$. A packed secret sharing is constructed via polynomial $F(y) \in \mathbb{F}[y]$ of degree at most d such that $F(\text{pos}_i) = \mathbf{x}_i$ for $i \in [\rho]$ and $F(\alpha_j) = x^j$, the j -th party's share, for all $j \in [n]$. Indeed, we will often make pos explicit and denote the packed secret sharing as $[\mathbf{x}||\text{pos}]_{d,d'}$.

An important property of both Shamir and packed secret sharing, following from the properties of polynomials, is *error-detection*. Here, it is well known that as long as the degree d satisfies $d \leq n - t - 1$ (i.e., the honest parties could reconstruct the whole sharing using just their shares), then the modification of any shares by corrupted parties can be detected.

Local homomorphisms. Both Shamir secret sharing and packed secret sharing enjoy the following homomorphisms: If the parties locally add their shares $[z]_d \leftarrow [x]_d + [y]_d$ (resp. $[\mathbf{z}||\text{pos}]_d \leftarrow [\mathbf{x}||\text{pos}]_d + [\mathbf{y}||\text{pos}]_d$), then they end up with degree- d shares of $z = x + y$ (resp. degree- d shares of $\mathbf{z} = \mathbf{x} + \mathbf{y}$). If the parties locally multiply their shares $[z]_{2d} \leftarrow [x]_d \cdot [y]_d$ (resp. $[\mathbf{z}||\text{pos}]_{2d} \leftarrow [\mathbf{x}||\text{pos}]_d \cdot [\mathbf{y}||\text{pos}]_d$), then they end up with degree- $2d$ shares of $z = x \cdot y$ (resp. degree- $2d$ shares of

$\mathbf{z} = \mathbf{x} * \mathbf{y}$, where $*$ denotes component-wise multiplication). As written above, it is important for packed sharings that adding or multiplying shares of two different secret vectors only works if $\mathbf{pos}$ is the same for both of the sharings. For Shamir secret sharing, the parties can also locally multiply a degree- d sharing by some constant $c \in \mathbb{F}$, $[y]_d \leftarrow c \cdot [x]_d$, to obtain a degree- d sharing of $y = c \cdot x$. On the other hand, for packed secret sharing, the parties can locally multiply a degree- d sharing by some constant vector $\mathbf{c} \in \mathbb{F}^\rho$, $[\mathbf{y}||\mathbf{pos}]_{d+\rho-1} \leftarrow \mathbf{c} * [\mathbf{x}||\mathbf{pos}]_d$, to obtain a degree- $(d + \rho - 1)$ packed sharing of $\mathbf{y} = \mathbf{c} * \mathbf{x}$. They do so by implicitly defining a (deterministic) degree- $(\rho - 1)$ sharing $[\mathbf{c}]_{\rho-1}$ of $\mathbf{c}$ (note it could be the case that $\rho \leq t$, but this is fine since $\mathbf{c}$ is not private).

Virtual Secret Sharings. To run our virtual MPC protocol, we will use packed secret sharings amongst the m virtual parties (with distinct evaluation elements $\alpha_1, \dots, \alpha_m$) in our protocol to pack $\rho = \Theta(m)$ secrets at a time. We call such a sharing an *outer* sharing. Furthermore, we will use standard Shamir secret sharing amongst the η real parties in a given committee to share such an outer share of the corresponding virtual party. We call such a sharing an *inner* sharing. Using an inner degree- d' sharing for each committee to share the shares of an outer degree- d sharing of length- ρ vectors is called a degree- (d, d') *virtual secret sharing*. In more detail, such a virtual secret sharing shares a secret vector $\mathbf{x} \in \mathbb{F}^\rho$, denoted as $[\mathbf{x}]_{d,d'}$. We denote the share of the j -th party of the i -th committee as $\mathbf{x}^{i,j}$. Often, we will make the positions $\mathbf{pos}$ of the secrets of the outer sharing explicit, and denote the virtual sharing as $[\mathbf{x}||\mathbf{pos}]_{d,d'}$. We will often use a default position vector, $\beta := (\beta_1, \dots, \beta_\rho)$. If we do not make the position vector $\mathbf{pos}$ explicit for a sharing $[\mathbf{x}]_{d,d'}$, then the positions of the secrets are given by β .

Observe that the parties can emulate local additions (resp. multiplications) $[\mathbf{x} + \mathbf{y}||\mathbf{pos}] \leftarrow [\mathbf{x}||\mathbf{pos}]_{d,d'} + [\mathbf{y}||\mathbf{pos}]_{d,d'}$ (resp. $[\mathbf{x} * \mathbf{y}||\mathbf{pos}]_{2d,2d'} \leftarrow [\mathbf{x}||\mathbf{pos}]_{d,d'} \cdot [\mathbf{y}||\mathbf{pos}]_{d,d'}$) by simply adding (resp. multiplying) their inner shares together. Indeed, the parties of each committee $\mathcal{C}_i$ share the i -th shares, $\mathbf{x}^i, \mathbf{y}^i$ of the outer sharing, through secret sharings $[\mathbf{x}^i]_{d'}, [\mathbf{y}^i]_{d'}$. If they add (resp. multiply) their shares together, then they obtain a sharing $[\mathbf{x}^i + \mathbf{y}^i]_{d'}$ (resp. $[\mathbf{x}^i \cdot \mathbf{y}^i]_{2d'}$) of the addition (resp. multiplication) of the outer shares. Since every committee does this, it indeed emulates the virtual parties adding (resp. multiplying) the outer shares together, resulting in the aforementioned virtual sharings. They can also therefore locally emulate multiplication by constant vector $\mathbf{c} \in \mathbb{F}^\rho$, $[\mathbf{c} * \mathbf{x}||\mathbf{pos}]_{d+\rho-1,d'} \leftarrow \mathbf{c} * [\mathbf{x}||\mathbf{pos}]_{d,d'}$.

Observe that with the committee sampling technique of Section 6, privacy will be preserved if $d \geq \tau + \rho - 1$ and $d' \geq \tau'$. Moreover, if $d' \leq \eta - \tau' - 1$, then error-detection can be used for inner sharings of honest committees, to either reconstruct the correct outer share, or abort. Using this, if $d + \rho - 1 \leq m - \tau - 1$, then error-detection can be used for the outer sharings, and thus the whole virtual sharing, even after local multiplication by constant vector, relying on the error-detected outer shares from honest committees. We will also require that $2d < m$ and $2d' < \eta$ so that reconstruction of degree- $(2d, 2d')$ virtual sharings can still happen (without error-detection).

Based on the above, throughout the paper we will set $d' = \tau'$, $d = \lfloor (m-1)/2 \rfloor$, and $\rho = d - \tau$ so that $\rho = \Theta(m) = \Theta(\sqrt{n})$, since δ is an arbitrarily small constant. With these settings, for *honest* committees, we clearly get privacy for degree d' inner sharings, error detection for degree d' inner sharings, and reconstruction for degree $2d'$ inner sharings. For outer sharings, we clearly get privacy for degree $\geq d$ outer sharings, error detection for degree $\leq d + \rho - 1$ outer sharings and reconstruction for degree $2d$ outer sharings. We will also use degree- $(m-1, \eta-1)$ virtual sharings; clearly privacy and reconstruction holds (as long as they are not multiplied by a constant vector), while error-detection of course does not hold.

5 MPC Building Blocks

In this section, we present some building blocks common to statistically-secure MPC protocols. As stated earlier, we will often times rely on the input parties to be *leaders* of the following protocols, especially for efficient randomness generation.

Virtual Random Sharings Generation. A core procedure in our protocol is π_{Rand} , which samples virtual random sharings by communicating $O(n|I|)$ elements, or $O(\sqrt{n}|I|)$ amortized per secret. We rely on the input parties to sample random sharings, which the parties sum up; if there is an honest input party, then the resulting sharing will be random.

Procedure 1: $\pi_{\text{Rand}}(\ell, \delta, \delta')$

Parameters: This procedure is parameterized by the number of desired output sharings ℓ , and degrees δ, δ' .

1. Each input party P_i first samples random vectors $\mathbf{r}_{i,j} \xleftarrow{\$} \mathbb{F}^\rho$ and random virtual sharings $\llbracket \mathbf{r}_{i,j} \rrbracket_{\delta, \delta'}$ for $j \in [\ell]$. They distribute the shares to the other parties.
2. All parties locally compute and output their shares of $\llbracket \mathbf{r}_j \rrbracket_{\delta, \delta'} \leftarrow \sum_{i \in I} \llbracket \mathbf{r}_{i,j} \rrbracket_{\delta, \delta'}$ for $j \in [\ell]$.^a

^a If we are in the secure erasure model, the parties also erase any randomness they sampled during the first step (besides their own inner shares)

On some occasions, we will write that the parties run π_{Rand} twice with different degrees to obtain two random sharings of the same secret vector $\mathbf{r}$ with these different degrees, $\llbracket \mathbf{r} \rrbracket_{d_1, d'_1}$ and $\llbracket \mathbf{r} \rrbracket_{d_2, d'_2}$. In this case, the honest input parties will use the same secret vectors $\mathbf{r}_{i,j}$ for the sharings of different degrees, $\llbracket \mathbf{r}_{i,j} \rrbracket_{d_1, d'_1}$ and $\llbracket \mathbf{r}_{i,j} \rrbracket_{d_2, d'_2}$. Note that the degree of the resulting random sharings might not be correct; We present a functionality $\mathcal{F}_{\text{RandVerif}}$, and protocol $\Pi_{\text{RandVerif}}$ instantiating it, in Section B.3 for sampling random sharings with correct degrees.

Other Building Blocks. Due to space restrictions, we defer the other common MPC building blocks we use to Appendix B. This includes $\mathcal{F}_{\text{RG}}$ which samples randomness for the parties with $\tilde{O}(n|I| + n^{1.5})$ communicated elements, $\mathcal{F}_{\text{DV}}$

which verifies the degree of virtual sharings with $\tilde{O}(n|I| + n^{1.5})$ communicated elements, $\mathcal{F}_{\text{RandVerif}}$, which samples virtual random sharings with the proper degree with $O(\sqrt{n}|I|)$ amortized communicated elements, and $\mathcal{F}_{\text{Input}}$ which allows each input party to share their with $O(n)$ communicated elements.

6 Committee Sampling

Our protocol makes use of committee sampling, in which the n parties are divided, with repetition, into a small number of sets of small size, for the purposes of executing certain sub-routines. Formally, we rely on samplers to construct our committees. Prior works [KKK⁺10, KS11] presented non-explicit constructions of samplers for Byzantine agreement. In our work, we present an explicit randomized construction of samplers that achieve the same properties except with negligible probability. We start by presenting a definition for samplers that is slightly adapted from prior works, but considers strict committee sizes.

Definition 1 (((α, β, γ) -Sampler). *For any $0 < \alpha < 1$, $0 < \beta < 1$ and $0 < \gamma < 1$, we say a function $H : [m] \rightarrow [n]^\eta$ is a (α, β, γ) -sampler, if for every set $S \subset [n]$ of size at most $|S| \leq \gamma \cdot n$, it is the case that the following holds*

$$\left| \left\{ x \in [m] \mid \frac{|H(x) \cap S|}{\eta} > \frac{|S|}{n} + \alpha \right\} \right| \leq \beta \cdot m$$

where $H(x)$ is a multi-set and $H(x) \cap S$ is the intersection of a multi-set and a set where duplicity is accounted.

In the context of MPC, the m committees will be defined by $H(x)$ for all $x \in [m]$ consisting of at most η parties each. The set S of γ -fraction parties may be viewed as the corrupted parties. Samplers guarantee that there are at most β -fraction of committees whose portion of corrupted parties is α -fraction more than the total fraction of corrupted parties, $|S|/n$.

We consider a randomly chosen H in the following way. For each $x \in [m]$, $H(x)$ is defined as choosing η items from $[n]$ uniformly at random with replacement. We show that such a randomly generated H will be a sampler except with negligible probability for appropriately selected parameters. We adapt the probabilistic method proof from [KKK⁺10] to show that this randomized construction of H is a sampler except with negligible probability.

Theorem 4. *For any choice of $0 < \alpha, \beta, \gamma < 1$, if $\eta \geq 6(n + \lambda)/(\alpha^2 \beta \gamma m)$ and $n + \lambda \geq m$, then the above randomized construction of H will be a (α, β, γ) -sampler except with probability $e^{-\lambda}$.*

Proof. First, we fix any choice of $S \subset [n]$. Consider any $x \in [m]$ where $H(x)$ is a uniformly random multi-set of η items from $[n]$. Note that the expected intersection size is $\mu = \mathbf{E}[|H(x) \cap S|] = \eta \cdot |S|/n$. By a Chernoff bound, we get

$$\Pr[|H(x) \cap S| \geq (1 + \alpha)\mu] \leq e^{-\frac{\alpha^2 \mu}{(2 + \alpha)}} = e^{-\frac{\alpha^2 \eta |S|}{(2 + \alpha)n}}.$$

Next, we fix any subset $R \subset [m]$ of size $|R| \geq \beta \cdot m$. We note the probability that all $x \in R$ satisfy $|H(x) \cap S| > (1 + \alpha)\mu$ is at most

$$\prod_{x \in R} \Pr[|H(x) \cap S| \geq (1 + \alpha)\mu] \leq e^{-\frac{\alpha^2 \eta |S|}{(2+\alpha)n} \cdot |R|} \leq e^{-\frac{\alpha^2 \eta |S|}{(2+\alpha)n} \cdot \beta m}.$$

Note that the choice of $H(x)$ for all $x \in [m]$ are independent. Therefore, the probabilities are also independent. Finally, we do a union bound over all at most 2^m choices of subset $R \subset [m]$ and all 2^n choices of subset $S \subset [n]$. Note, it suffices to only consider subsets S of exactly size $\gamma \cdot n$ as if there exists a subset of smaller size satisfying this event, it is easy to see there will always be at least one subset of size exactly $\gamma \cdot n$ also satisfying the event. Altogether, the probability that H is not a (α, β, γ) -sampler is at most $2^n \cdot 2^m \cdot e^{-\frac{\alpha^2 \eta |S| \beta m}{(2+\alpha)n}} \leq e^{-\frac{\alpha^2 \eta \gamma \beta m}{3} + n + m}$ where we used the fact that $2 + \alpha < 3$ and $|S| = \gamma \cdot n$. In other words, we need that the following inequality is satisfied: $-(\alpha^2 \eta \gamma \beta m)/3 + n + m \leq -\lambda$. We can rearrange to obtain the desired bound on d as follows:

$$-(\alpha^2 \eta \gamma \beta m)/3 + n + m \leq -\lambda \iff 1 + (\lambda + n)/m \leq (\alpha^2 \eta \gamma \beta)/3.$$

Note, it suffices to set $\eta \geq 6(\lambda + n)/(\alpha^2 \beta \gamma m)$ to satisfy the above inequality assuming $\lambda + n \geq m$ to complete the proof. $\square$

Committee Sampling Protocol. We formally describe the ideal functionality informing each party about properly randomly sampled committees along with the identities of other parties that belong to the same committee with the party.

Functionality 2: $\mathcal{F}_{\text{CSamp}}$

1. $\mathcal{F}_{\text{CSamp}}$ independently samples $m = \tilde{O}(\sqrt{n})$ subsets of $[n]$: $\mathcal{C}_1, \dots, \mathcal{C}_m$, where for all $i \in [m]$, $\mathcal{C}_i$ is generated by sampling each $P \in [n]$ into $\mathcal{C}_i$ independently w.p. $\frac{1}{\sqrt{n}}$. $\mathcal{F}_{\text{RG}}$ sends $\mathcal{C}_1, \dots, \mathcal{C}_m$ to the adversary.
2. $\mathcal{F}_{\text{CSamp}}$ waits for a message from the adversary, if its **abort**, the functionality aborts. Else, if the adversary chooses to corrupt all input parties, the functionality allows the adversary to dictate the outputs of each party as it wants.
3. Else, for each party P , $\mathcal{F}_{\text{CSamp}}$ sends to P the set $\{\mathcal{C}_i \mid i \in [m], P \in \mathcal{C}_i\}$.

We instantiate $\mathcal{F}_{\text{CSamp}}$ via the protocol Π_{CSamp} , which we describe next. Intuitively, we sample committees by having the input parties generate a uniformly random seed string, from which the committees are parsed. The challenge, of course, is that the input parties may have a dishonest majority. To overcome this, we let input parties secret share their seed contributions to all parties, that add up the shares on their own, and then reconstruct back to the input parties.

Protocol 2: Π_{CSamp}

Parameters: Let $k = \Theta(n)$, $k < \epsilon n$ be a *packing parameter*. Let $\mathbb{K}$ be a field such that $\log |\mathbb{K}| = \Theta(\log n)$. We will use degree- $(t + k - 1)$ packed sharings over $\mathbb{K}$, denoted as $[\mathbf{x}]_{t+k-1}^{\mathbb{K}}$.

1. Each input party $P_i \in I$ generates and distributes random packed sharing $[\mathbf{x}_i]_{t+k-1}^{\mathbb{K}}$ to all n parties.
2. The parties compute $[\mathbf{x}]_{t+k-1}^{\mathbb{K}} \leftarrow \sum_{i: P_i \in I} [\mathbf{x}_i]_{t+k-1}^{\mathbb{K}}$ and reconstruct it to each input party $P_i \in I$.
3. Each input party $P_i \in I$ reconstructs $\mathbf{x}$ (aborting if error-detection fails) and interprets it as a length- $(n \log n)$ bit string, S .
4. Each input party $P_i \in I$ parses S as committee assignments $\mathcal{C}_1, \dots, \mathcal{C}_m$ for $m = \tilde{O}(\sqrt{n})$, where $|\mathcal{C}_i| = \tilde{O}(\sqrt{n})$.
5. Let P_ℓ be some designated party in I . P_ℓ sends, for each committee $\mathcal{C}_i$, and each party $P_{i,j}$ in that committee, all members of $\mathcal{C}_i$.
6. Afterwards, all other input parties P_i locally sample seed s_i for a universal hash function $\mathcal{H}$. Each of them then sends to all parties P the *hash* (based on s_i) of the committee definitions in which P is a member, along with the seed s_i itself.
7. All parties record their received committees, and if they detected any inconsistency between the received committees and the hashes, **abort**.

We next prove that Π_{CSamp} securely instantiates the functionality $\mathcal{F}_{\text{CSamp}}$. First note that the communication complexity of Π_{CSamp} is $O(n^{1.5} + |I|n\kappa)$ bits. First, note that the cost of distributing shares of the random seeds and sending them back to the input parties requires $O(n \cdot |I| \cdot \log |\mathbb{K}|) = O(n \cdot |I| \cdot \log n)$ bits of communication. Next, note that there are $\tilde{O}(\sqrt{n})$ committees, each of size $\tilde{O}(\sqrt{n})$, and from Theorem 4 we have that with high probability, each party P is in at most $\tilde{O}(1)$ committees. As such, with high probability, P_ℓ sends to each party at most $\tilde{O}(\sqrt{n})$ bits, amounting in total to $O(n^{1.5})$ bits of communication. Finally, the rest of the input parties send to all other parties a hash and a random seed, both of length $O(\kappa)$, amounting to $O(|I|n\kappa)$ communication.

We show that if at least one input party remains honest throughout the execution of Π_{CSamp} , then the resulting committees constitute a sampler.

Lemma 1. *Suppose that in an execution of Π_{CSamp} , there exists an input party $p \in I$ that remains honest for the entirety of the execution. Then, either honest parties abort, or the resulting outputs held by every honest party constitute a sampler for an appropriate choice of constants β, α , and for $\gamma = \frac{1}{2} - \epsilon$.*

Proof. Let $P_i \in I$ be an input party and suppose it remains honest for the entire execution of Π_{CSamp} . In particular, it implies that it correctly samples a sharing $[\mathbf{x}_i]_{t+k-1}^{\mathbb{K}}$ in step 1 of the protocol and shares it correctly. This implies that if P_i didn't detect any inconsistency in the added sharing $[\mathbf{x}]_{t+k-1}^{\mathbb{K}}$, then S is a uniformly random bit string of proper length. Such a string can be then used by

P_i to sample a (α, β) -sampler for arbitrarily small constants α, β ⁶ according to the distribution described in Theorem 4. From here there are two cases, either P_ℓ is honest and then we can assume that $P_i = P_\ell$, from which we have that all parties obtain a correct committee assignment based on S , and the adversary can only cause aborts by sending inconsistent hashes.

Otherwise, P_ℓ is corrupt, and in that case, after P_ℓ distributes the committee assignments, P_i then hashes the appropriate information (based on S) required for each party and sends it to each party. The only way for an adversarial party to fool the honest parties is if P_ℓ is corrupt and gets lucky by guessing the universal hash function seed of P_i , an event that occurs with negligible probability. Note that Π_{CSamp} dictates that P_ℓ distribute the committee assignments *prior* to the rest of the parties sending the hashes and seeds, and thus the adversary, in case P_ℓ is corrupted, can not predict beforehand the contents of the seed of P_i . This implies that if P_ℓ attempts to distribute a committee assignment that does not correspond to S , with high probability P_i 's hash will be inconsistent with it and would cause the receiving party to abort, as required. $\square$

Lemma 2. Π_{CSamp} UC-realizes $\mathcal{F}_{\text{CSamp}}$.

Proof. Note that the privacy requirements are vacuously satisfied as $\mathcal{F}_{\text{CSamp}}$ does not involve any input values. Note further that the functionality $\mathcal{F}_{\text{CSamp}}$ sends all committee data to the adversary, and so the simulation is straightforward by having the simulator first send to the adversary corrupted parties' random shares, and then honest parties' shares consistent with those and the committee assignments. Lemma 1 guarantees that so long as not all input parties are corrupted, the real world and the ideal world views of honest parties are the same. If all input parties are corrupted, then the functionality allows the adversary complete control over the inputs, and so any real world behavior is simulatable. $\square$

7 Sharing Transformation

We now show how to perform sharing transformations [GPS22] on virtual sharings. Such a sharing transformation allows the parties to transform a virtual sharing $\llbracket \mathbf{x} \mid \mathbf{pos} \rrbracket_{m-1, \eta-1}$ to a virtual sharing $\llbracket \mathbf{f}(\mathbf{x}) \mid \mathbf{pos}' \rrbracket_{d, d'}$, where $f : \mathbb{F}^\rho \rightarrow \mathbb{F}^\rho$ is a linear map and $\mathbf{pos}, \mathbf{pos}'$ are position vectors. Such a sharing transformation will be crucial in our eventual MPC protocol. Indeed, since each virtual sharing $\llbracket \mathbf{x} \rrbracket_{d, d'}$ stores a vector of secrets, the parties need to ensure that for every layer of the circuit, and each group of, e.g., ρ addition gates at that layer, there are sharings $\llbracket \mathbf{x} \rrbracket_{d, d'}$ and $\llbracket \mathbf{y} \rrbracket_{d, d'}$ prepared corresponding to exactly the inputs to those addition gates, so that the parties can compute $\llbracket \mathbf{x} + \mathbf{y} \rrbracket_{d, d'} \leftarrow \llbracket \mathbf{x} \rrbracket_{d, d'} + \llbracket \mathbf{y} \rrbracket_{d, d'}$. Since such sharings $\llbracket \mathbf{x} \rrbracket_{d, d'}$ and $\llbracket \mathbf{y} \rrbracket_{d, d'}$ may not be automatically generated based on the previous layers' gates, the parties must utilize sharing transformations to prepare them. This is called *network routing*; see [GPS22] for more discussion.

⁶ We always assume throughout the protocol a sufficiently small constant choice for α, β . This incurs a constant multiplicative factor in the communication complexity.

The intuition for performing a sharing transformation is as follows. The parties will prepare random sharings $\llbracket \mathbf{r} \parallel \mathbf{pos} \rrbracket_{m-1, \eta-1}$ and $\llbracket \mathbf{f}(\mathbf{r}) \parallel \mathbf{pos}' \rrbracket_{d, d'}$ that we show in Section 7. Then the parties will compute $\llbracket \mathbf{e} \parallel \mathbf{pos} \rrbracket_{m-1, \eta-1} \leftarrow \llbracket \mathbf{x} \parallel \mathbf{pos} \rrbracket_{m-1, \eta-1} + \llbracket \mathbf{r} \parallel \mathbf{pos} \rrbracket_{m-1, \eta-1}$ and open $\llbracket \mathbf{e} \parallel \mathbf{pos} \rrbracket_{m-1, \eta-1}$ to one another. Finally, the parties compute $f(\mathbf{e})$ and the canonical virtual sharing $\llbracket \mathbf{f}(\mathbf{e}) \parallel \mathbf{pos}' \rrbracket_{\rho-1, 0}$ and output $\llbracket \mathbf{f}(\mathbf{x}) \parallel \mathbf{pos}' \rrbracket_{d, d'} \leftarrow \llbracket \mathbf{f}(\mathbf{e}) \parallel \mathbf{pos}' \rrbracket_{\rho-1, 0} - \llbracket \mathbf{f}(\mathbf{r}) \parallel \mathbf{pos}' \rrbracket_{d, d'}$. Indeed, since f is linear $f(\mathbf{x}) = f(\mathbf{e}) - f(\mathbf{r})$.

Random Sharings for Different Linear Secret Sharing Schemes. There may be several sharing transformations that the parties have to perform throughout the MPC for network routing, without many duplicates. [GPS22] showed how to generate all such pairs of random sharings $\llbracket \mathbf{r} \parallel \mathbf{pos} \rrbracket_{m-1, \eta-1}, \llbracket \mathbf{f}(\mathbf{r}) \parallel \mathbf{pos}' \rrbracket_{d, d'}$ with low overhead. We translate their ideas to our setting of virtual MPC here.

The functionality. We first present the functionality that generates a single such pair of random sharings. It allows the adversary to inject an additive error $\delta \in \mathbb{F}^\rho$ to the secret vector of the latter sharing. Observe that for the $\llbracket \mathbf{s} \rrbracket_{m-1, \eta-1}$ sharing, each committee $\mathcal{C}_i$'s outer share is determined by a linear combination of the inner shares: $\mathbf{s}^i = \sum_{j=1}^{\eta} c_j \cdot \mathbf{s}^{ij}$. Furthermore, each secret $\mathbf{s}_k$ is determined by a linear combination of the outer shares $\mathbf{s}_k = \sum_{i=1}^m c_{i,k} \cdot \mathbf{s}^i$. Therefore, each secret is indeed a linear combination of all of the inner shares. We utilize this fact in $\mathcal{F}_{\text{MapRand}}$ below, by ensuring that the terms involving the honest parties' shares added to the value Δ_k representing the sum of the terms involving the corrupted parties' shares indeed sum to $\mathbf{s}_k$ for each $k \in [\rho]$.

Functionality 3: $\mathcal{F}_{\text{MapRand}}$

Parameters: This functionality is parameterized by a linear map f and position vectors $\mathbf{pos}, \mathbf{pos}'$.

1. $\mathcal{F}_{\text{MapRand}}$ first may receive **abort** from the adversary; if so, it outputs **abort** to the honest parties.
2. Otherwise, $\mathcal{F}_{\text{MapRand}}$ receives from the adversary the set of outer shares $\{\mathbf{s}^i\}_{i \in \mathcal{T}_{\text{virt}}}$ of the corrupted virtual parties.
3. Also, for each committee $\mathcal{C}_i$ in which $|\mathcal{T}_i| \leq d'$, $\mathcal{F}_{\text{MapRand}}$ receives from the adversary the set of corrupted parties' corresponding inner shares $\{\mathbf{s}^{ij}\}_{j \in \mathcal{T}_i}$.
4. Additionally, for each committee $\mathcal{C}_i$ in which $|\mathcal{T}_i| > d'$, $\mathcal{F}_{\text{MapRand}}$ receives the honest parties' corresponding inner share $\{\mathbf{s}^{ij}\}_{j \in \mathcal{H}_i}$.
5. The adversary also sends $\mathcal{F}_{\text{MapRand}}$ vectors ε and Δ .
6. $\mathcal{F}_{\text{MapRand}}$ samples a random $\mathbf{s}$ and random virtual sharing $\llbracket \mathbf{f}(\mathbf{s}) + \varepsilon \parallel \mathbf{pos}' \rrbracket_{d, d'}$ consistent with the shares received above, as well as random honest parties' shares conditioned on combining with the vector Δ to result in $\llbracket \mathbf{s} \parallel \mathbf{pos} \rrbracket_{m-1, \eta-1}$. $\mathcal{F}_{\text{MapRand}}$ then distributes to the honest parties their shares.

Sub-Procedures for instantiating $\mathcal{F}_{\text{MapRand}}$. We now present two sub-procedures for instantiating $\mathcal{F}_{\text{MapRand}}$. We start with $\pi_{\text{CommRand}}[\mathcal{C}_i]$, in which we again leverage the input parties to sample and share the same random vector $\mathbf{r}$ amongst both (i) the virtual parties and (ii) just the real parties of committee $\mathcal{C}_i$.

Procedure 3: $\pi_{\text{CommRand}}[\mathcal{C}_i, \delta]$

Parameters: This protocol is parameterized by a specific committee $\mathcal{C}_i \subseteq [n]$ and a degree δ .

1. Each input party P_j first samples random vectors $\mathbf{r}_j \xleftarrow{\$} \mathbb{F}^\rho$, random virtual sharings $[\mathbf{r}_j]_{d+\rho-1, d'}$ and for each $k \in [\rho]$, random sharings for committee $\mathcal{C}_i$, $[\mathbf{r}_{j,k}]_\delta$. They distribute the shares to the other parties.
2. All parties locally compute $[\mathbf{r}]_{d+\rho-1, d'} \leftarrow \sum_{j \in I} [\mathbf{r}_j]_{d+\rho-1, d'}$, and the parties in committee $\mathcal{C}_i$ locally compute $[\mathbf{r}_k]_\delta \leftarrow \sum_{j \in I} [\mathbf{r}_{j,k}]_\delta$.
3. The parties output their shares.

The above procedure $\pi_{\text{CommRand}}[\mathcal{C}_i]$ will allow the parties of just committee $\mathcal{C}_i$ to obtain secret sharings of the elements of a vector $\mathbf{x}$, originally shared amongst the virtual parties. This is done in the following procedure $\pi_{\text{CommOpen}}[\mathcal{C}_i]$.

Procedure 4: $\pi_{\text{CommOpen}}[\mathcal{C}_i, \delta]$

Parameters: This protocol is parameterized by a specific committee $\mathcal{C}_i \subseteq [n]$ and a degree δ .

1. Let $[\mathbf{x}]_{d+\rho-1, d'}$ be the input packed virtual sharing.
2. The parties invoke $\pi_{\text{CommRand}}[\mathcal{C}_i]$ to obtain $[\mathbf{r}]_{d+\rho-1, d'}$ and $[\mathbf{r}_j]_\delta$ shared amongst the parties of committee $\mathcal{C}_i$, for $j \in [\rho]$.
3. The parties compute $[\mathbf{e}]_{d+\rho-1, d'} \leftarrow [\mathbf{x}]_{d+\rho-1, d'} + [\mathbf{r}]_{d+\rho-1, d'}$ and open it to party P_1 .
4. Party P_1 reconstructs $\mathbf{e} := (e_1, \dots, e_\rho)$ and sends it to the parties of committee $\mathcal{C}_i$.
5. Finally, the parties of committee $\mathcal{C}_i$ compute and output $[\mathbf{x}_j]_\delta \leftarrow \mathbf{e}_j - [\mathbf{r}_j]_\delta$ for $j \in [\rho]$.

Note: it could be the case that $[\mathbf{r}]_{d+\rho-1, d'}$ and the sharings $[\mathbf{r}_k]_\delta$ for $k \in [\rho]$ share different secret vectors. Moreover, P_1 could be corrupted and distribute the incorrect vector $\mathbf{e}$ to the honest parties. This is captured in $\mathcal{F}_{\text{MapRand}}$ by the fact the adversary can input error vector δ . The communication complexity of π_{CommOpen} and π_{CommRand} is $O(n|I|)$ elements for a single committee $\mathcal{C}_i$.

Instantiating $\mathcal{F}_{\text{MapRand}}$. Finally, we present protocol Π_{MapRand} that instantiates $\mathcal{F}_{\text{MapRand}}$. The intuition for this protocol comes from [GPS22].

Let $[\mathbf{s}||\mathbf{pos}]_{m-1, \eta-1}, [\mathbf{f}(\mathbf{s})||\mathbf{pos}']_{d, d'}$ be a random pair that we want to generate for a given Sharing Transformation. Focusing on just $[\mathbf{s}||\mathbf{pos}]_{m-1, \eta-1}$ for now, its outer sharing $[\mathbf{s}||\mathbf{pos}]_{m-1}$ is represented by a degree- $(m-1)$ poly-

mial $f(y)$, as explained in Section 4. A procedure to sample $[s||\text{pos}]_{m-1}$ is as follows: Sample ρ random elements $s_1, \dots, s_\rho$ and $m - \rho$ additional random elements $r_1, \dots, r_{m-\rho}$ from $\mathbb{F}$. Then, perform lagrange interpolation to compute $f(y)$ and thus the shares of the virtual parties, $f(j)$. More specifically, the share of the j -th party can be written as the linear operation on $r_1, \dots, r_m$, $s^j := f(j) \leftarrow \sum_{i=1}^{\rho} c_i \cdot s_i + \sum_{i=\rho+1}^m c_i \cdot r_{i-\rho}$, where each c_i is an appropriate lagrange coefficient. The shares of $[f(s)||\text{pos'}]_d$ is sampled based on linear interpolation of $s_1, \dots, s_\rho$ and additional $d + 1 - \rho$ random elements, $u_1, \dots, u_{d+1-\rho}$.

Therefore, if we sample random virtual sharings $\llbracket s_1 \rrbracket_{d,d'}, \dots, \llbracket s_\rho \rrbracket_{d,d'}, \llbracket r_1 \rrbracket_{d,d'}, \dots, \llbracket r_{m-\rho} \rrbracket_{d,d'}$, and $\llbracket u_1 \rrbracket_{d,d'}, \dots, \llbracket u_{d+1-\rho} \rrbracket_{d,d'}$, then the parties can compute *virtual sharings of the shares* of the random pair $[s||\text{pos}]_{m-1}, [f(s)||\text{pos'}]_d$, since the above operations are linear. Indeed, since each s_i, r_i, u_i are *vectors*, we can actually sample virtual sharings of shares of many such random pairs at once, for better efficiency. The coefficients c_i of each interpolation may be different (if the position vectors pos and pos' are different for each random pair), but this can be handled since we can actually perform *multiplication by constant* on our virtual sharings.

For simplicity, in Π_{MapRand} , we will denote the originally sampled random vectors as $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_{m+d+1-\rho}$. Thus, letting $\ell = m + d + 1 - \rho$ the coefficient vectors for indices $i \in [\ell]$ will either represent the relevant Lagrange coefficient vectors of the corresponding interpolations, or $\mathbf{0}$. Since the virtual sharings of shares of random pairs are for outer shares, we use π_{CommOpen} to open to each committee their virtual party's share (itself secret shared amongst the committee).

Protocol 5: Π_{MapRand}

Parameters: This protocol is parameterized by linear maps $f_1, \dots, f_\rho : \mathbb{F}^\rho \rightarrow \mathbb{F}^\rho$, position vectors $(\text{pos}_1, \text{pos}'_1), \dots, (\text{pos}_\rho, \text{pos}'_\rho)$, and integer $\ell = m + d + 1 - \rho$. Let f be the concatenation of $f_1, \dots, f_\rho$.

1. For $j \in [\ell]$, the parties invoke $\mathcal{F}_{\text{RandVerif}}$ to obtain sharings $\llbracket \mathbf{r}_j \rrbracket_{d,d'}$.
2. For $j \in [\ell]$, the parties compute and output $\llbracket \mathbf{s}^j \rrbracket_{d+\rho-1,d'} \leftarrow \sum_{i=1}^{\ell} c_{i,1} * \llbracket \mathbf{r}_j \rrbracket_{d,d'}$ and $\llbracket \mathbf{f}(\mathbf{s})^j \rrbracket_{d+\rho-1,d'} \leftarrow \sum_{i=1}^{\ell} c_{i,2} * \llbracket \mathbf{r}_j \rrbracket_{d,d'}$.
3. For $j \in [\ell]$, the parties invoke $\pi_{\text{CommOpen}}[C_j, \eta - 1](\llbracket \mathbf{s}^j \rrbracket_{d+\rho-1,d'})$ and $\pi_{\text{CommOpen}}[C_j, d'](\llbracket \mathbf{f}(\mathbf{s})^j \rrbracket_{d+\rho-1,d'})$ so that the parties of committee C_j receive $\llbracket \mathbf{s}_k^j \rrbracket_{\eta-1}$ and $\llbracket \mathbf{f}_k(\mathbf{s}_k)^j \rrbracket_{d'}$ for $k \in [\rho]$.
4. The parties interpret the above as $(\llbracket \mathbf{s}_k || \text{pos}_k \rrbracket_{m-1,\eta-1}, \llbracket \mathbf{f}(\mathbf{s}_k) || \text{pos}'_k \rrbracket_{d,d'})$ for $k \in [\rho]$.
5. The parties run $\mathcal{F}_{\text{DV}}$ on $\llbracket \mathbf{f}(\mathbf{s}_k) || \text{pos}'_k \rrbracket_{d,d'}$ for $k \in [\rho]$; if it does not abort, they output the sharings.

Observe that $\ell = O(m) = O(\sqrt{n})$. Thus, the communication complexity of Π_{MapRand} is $O(n^{1.5}|I|)$ for the invocations of $\mathcal{F}_{\text{RandVerif}}$, $O(n^{1.5}|I|)$ elements for the invocations of π_{CommOpen} and $O(n^{1.5})$ elements for the invocation of $\mathcal{F}_{\text{DV}}$. Since we end up with $\Theta(\sqrt{n})$ sharings of $\Theta(\sqrt{n})$ underlying secrets, the communication is

$O(\sqrt{n} \cdot |I|)$ elements amortized. Π_{MapRand} can be run once at the beginning of the entire main MPC protocol, Π_{Main} . We prove the below lemma in Appendix D.1.

Lemma 3. Π_{MapRand} UC-realizes $\mathcal{F}_{\text{MapRand}}$ in the $(\mathcal{F}_{\text{RandVerif}}, \mathcal{F}_{\text{DV}})$ -hybrid model.

Sharing Transformation Procedure. With $\mathcal{F}_{\text{MapRand}}$, we instantiate the sharing transformation procedure $\pi_{\text{SharingTrans}}$. The communication complexity of $\pi_{\text{SharingTrans}}$ is $O(\sqrt{n} \cdot |I|)$ elements from $\mathcal{F}_{\text{MapRand}}$ and $O(n)$ total for the rest.

Procedure 6: $\pi_{\text{SharingTrans}}[f]$

Parameters: This protocol is parameterized by a linear map f and position vectors pos and pos' .

1. Let $\llbracket x || \text{pos} \rrbracket_{m-1, \eta-1}$ be the input packed virtual sharing.
2. The parties invoke $\mathcal{F}_{\text{MapRand}}(f, \text{pos}, \text{pos}')$ to obtain random packed virtual sharings $\llbracket r || \text{pos} \rrbracket_{m-1, \eta-1}$ and $\llbracket f(r) || \text{pos}' \rrbracket_{d, d'}$.
3. The parties compute $\llbracket e || \text{pos} \rrbracket_{m-1, \eta-1} \leftarrow \llbracket x || \text{pos} \rrbracket_{m-1, \eta-1} + \llbracket r || \text{pos} \rrbracket_{m-1, \eta-1}$ and open their shares to party P_1 .
4. Party P_1 reconstructs e , then computes $\llbracket f(e) || \text{pos}' \rrbracket_{\rho-1, 0}$, and distributes the shares to all parties.^a
5. Finally, the parties compute and output $\llbracket f(x) || \text{pos}' \rrbracket_{d, d'} \leftarrow \llbracket f(e) || \text{pos}' \rrbracket_{\rho-1, 0} - \llbracket f(r) || \text{pos}' \rrbracket_{d, d'}$.

^a The degrees of these sharings will be checked at the end of Π_{Main} .

8 Output Sampling

For the output phase of our protocol, we employ a randomized communication procedure for which we require the following lemma.

Lemma 4. Consider m' input buckets with z balls in each, and consider n output bins that are initially empty. Consider the process of assigning (ℓ copies) of each of the balls in each of the input buckets to ℓ of the output bins independently and uniformly at random, with repetitions. Then as long as $z \cdot \ell \geq n(\log m' + \log n + \lambda)$, it holds with probability at least $1 - 2^{-\lambda}$ that all outputs bins contain at least one ball from each of the m' input buckets.

Proof. We first note that for a given ball and a given bin, the probability that a single copy of the ball does not hit the bin is $1 - \frac{1}{n}$, the probability that all ℓ copies of the ball do not end up in the bin is $(1 - \frac{1}{n})^\ell$. Finally, the probability that none of the copies of any of the balls in a given bucket hit the bin is $(1 - \frac{1}{n})^{\ell \cdot z}$. Denote this event by E . Employing $(1 - x) \leq e^{-x}$, we get that $\Pr[E] \leq e^{-\frac{z \cdot \ell}{n}}$. A union bound over all m' buckets and n bins gives us that the probability that there exists a bucket such that no ball from that bucket landed in a particular bin is at most $m' \cdot n \cdot e^{-\frac{z \cdot \ell}{n}}$. We stipulate $m' \cdot n \cdot e^{-\frac{z \cdot \ell}{n}} \leq 2^{-\lambda}$. We thus have that $\log m' + \log n - \frac{z \cdot \ell}{n} \leq -\lambda$ holding $n(\log m' + \log n + \lambda) \leq z \cdot \ell$, as required. $\square$

For MPC, the input buckets are the $m' = (1 - \delta)$ “good” committees and the n outputs bins represent the parties and whether they receive an output from an honest party within each good committee. In the proof of Lemma 4, we consider that each ball is placed independently and uniformly at random. Indeed, this does not need to be true for outputs distributed from maliciously corrupted parties. Instead, we only consider the z balls as the z honest parties within each good committee, which is guaranteed by our samplers by Theorem 4.

Functionality 4: $\mathcal{F}_{\text{Open}}$

1. Let $\llbracket \mathbf{y} \rrbracket_{d,d'}$ be the sharing to be output.
2. $\mathcal{F}_{\text{Open}}$ receives from honest parties their shares and reconstructs $\llbracket \mathbf{y} \rrbracket_{d,d'}$.
3. $\mathcal{F}_{\text{Open}}$ sends $\llbracket \mathbf{y} \rrbracket_{d,d'}$ to adversary $\mathcal{A}$, and waits for $\mathcal{H}_{\text{abort}} \subseteq \mathcal{H}$ from $\mathcal{A}$.
4. Finally, $\mathcal{F}_{\text{Open}}$ sends **abort** to all honest parties in $\mathcal{H}_{\text{abort}}$ and $\mathbf{y}$ to all other honest parties $\mathcal{H} \setminus \mathcal{H}_{\text{abort}}$.

Protocol 7: Π_{Open}

1. Let $\llbracket \mathbf{y} \rrbracket_{d,d'}$ be the sharing to be output.
2. For each committee $\mathcal{C}_i$:
 - (a) The parties $P_{i,j}$ in that committee reconstruct to each other the outer share $\mathbf{y}^i$.
 - (b) Each such party $P_{i,j}$ locally samples a subset $S_{i,j}$ of parties by including every $P \in [n]$ into $S_{i,j}$ with probability $\frac{O(\log n)}{\sqrt{n}}$.
 - (c) Each $P_{i,j}$ sends reconstructed outer share $\mathbf{y}^i$ to all parties in $S_{i,j}$.
3. All parties P_k , upon receiving outer shares, if they did not receive enough shares to reconstruct output, then abort. If any inconsistency in received shares is detected, then abort. Else, reconstruct output $\mathbf{y}$.
4. All parties P_k **do the following even if aborted, with $\mathbf{y} = \text{abort}$** : Locally sample subset S_k of parties by including every $P \in [n]$ into S_k with probability $\frac{O(\log n)}{\sqrt{n}}$. P_k sends output $\mathbf{y}$ to all parties in S_k .
5. If a party receives abort or more than one claimed output, it aborts. Else, outputs $\mathbf{y}$.

Π_{Open} communicates $\tilde{O}(n^{1.5})$ elements as each honest party sends messages to at most $\tilde{O}(\sqrt{n})$ parties. This is $\tilde{O}(n)$ amortized per output. We will prove the following results.

Lemma 5. *At the end of Π_{Open} , with overwhelming probability there exists a value $\mathbf{y}$ such that all honest parties either abort, or output $\mathbf{y}$.*

Proof. Consider the state of the network after step 3 of Π_{Open} , in which every honest party P either has some output $\mathbf{y}_p$ or abort (which we denote by $\perp$). Note that we can partition into two cases: either there exists a value $\mathbf{y}$ held by a majority of the honest parties (i.e. at least $\frac{n}{4}$ parties), or not. If so, then consider

a party P' that does not hold $\mathbf{y}$. The probability that it does not hear from a particular party holding $\mathbf{y}$ in step 4 of Π_{Open} is $1 - \frac{O(\log n)}{\sqrt{n}}$. The probability it does not hear from *all* parties holding $\mathbf{y}$ is thus upper bounded by $(1 - \frac{O(\log n)}{\sqrt{n}})^{n/4}$, which by $(1 - x) \leq e^{-x}$, can be further upper bounded by $e^{-\tilde{\Omega}(\sqrt{n})}$. A union bound over all honest parties give that the probability that any of them does not hear from any honest party holding $\mathbf{y}$ is at most $n \cdot e^{-\tilde{\Omega}(\sqrt{n})}$. We thus get the probability *all* honest parties hear $\mathbf{y}$ from at least one other party is at least $1 - ne^{-\tilde{\Omega}(\sqrt{n})}$. Thus we have that w.o.p, all honest party either output $\mathbf{y}$ or abort.

The other case is that in which no value is held by a majority of the honest parties. Note however that the behavior of Π_{Open} has a party abort if it hears *any* other value from the one it holds. An identical analysis to above yields that with overwhelming probability, every honest party hears of a value not equal to the one it holds, and so all honest parties abort. $\square$

Lemma 6. *With overwhelming probability, every honest P hears a share from at least one honest party in every good committee C_i in step 2 of Π_{Open} .*

Proof. The proof of this lemma follows from Lemma 4. m' stands for the number of good committees, z stands for the number of good parties in each good committee, and n is the number of parties. Notice that the randomized process taking place in step 2 of Π_{Open} is identical to the one described in Lemma 4. We have that the number of good committees is $m' = \Omega(\sqrt{n})$, in each good committee, more than half parties are honest, and thus $z = \Omega(\sqrt{n})$. The lemma now follows from an application of 4. $\square$

Lemma 7. Π_{Open} UC-realizes $\mathcal{F}_{\text{Open}}$.

Proof. The simulation is straightforward as the functionality $\mathcal{F}_{\text{Open}}$ provides the adversary with all collected shares, and thus there are no privacy requirements. By Lemma 6, and Lemma 5, what the parties output in the ideal world (possibly abort) is identical to the output of the real world. $\square$

9 Full MPC Protocol

In this section, we will construct the remaining tools needed for the full MPC protocol, and then the full protocol itself, Π_{Main} . Throughout the protocol, we will associate with each wire w in the circuit C its own *position element* $p_w \in \mathbb{F}$.

We use the technique of [CGH⁺18], in which the parties sample a random sharing $\llbracket \mathbf{r} \rrbracket_{d,d'}$ at the beginning of the protocol (before the input phase), and in addition to computing a sharing of every wire in the circuit, $\llbracket \mathbf{w} \rrbracket_{d,d'}$, they also compute a version that is randomized by $\mathbf{r}$, $\llbracket \mathbf{r} \cdot \mathbf{w} \rrbracket_{d,d'}$. At the end of the protocol they check that indeed $\llbracket \mathbf{r} \cdot \mathbf{w} \rrbracket_{d,d'}$ shares the secret of $\llbracket \mathbf{w} \rrbracket_{d,d'}$, multiplied by $\mathbf{r}$. The intuition here is that given some multiplication of $\llbracket \mathbf{x} \rrbracket_{d,d'}$ and $\llbracket \mathbf{y} \rrbracket_{d,d'}$ resulting in $\llbracket \mathbf{z} \rrbracket_{d,d'}$, if the adversary causes the output sharing to share $\mathbf{z} = \mathbf{x} \cdot \mathbf{y} + \delta$, for some $\delta \neq 0$, then the adversary needs to ensure that $\llbracket \mathbf{r} \cdot \mathbf{z} \rrbracket$ indeed shares $\mathbf{r} \cdot (\mathbf{x} \cdot \mathbf{y} + \delta)$.

However, since $\mathbf{r}$ is random and unknown to the adversary, the adversary would have to guess $\mathbf{r}$ to do so, which can only happen with negligible probability.

Due to space limitations, we present the following procedures, which follow straightforwardly from standard techniques, [CGH⁺18], and [GPS22], in Appendix C: π_{Mult} for performing multiplications [DN07], $\pi_{\text{CheckMult}}$ for checking multiplications at the end of the protocol [CGH⁺18], π_{Network} for routing packed wire secret sharings [GPS22].

Complete MPC Protocol. We are now ready to present our MPC protocol. The parties proceed by first sampling random committees with $\mathcal{F}_{\text{CSamp}}$. Then, they use $\mathcal{F}_{\text{RandVerif}}$ to sample the random challenge r , with which the parties, for every batch of wires $\mathbf{x}$, compute both $\llbracket \mathbf{x} \rrbracket_{d,d'}$ and $\llbracket \mathbf{r} \cdot \mathbf{x} \rrbracket_{d,d'}$. The input parties then provide their input sharings $\llbracket \mathbf{x}_i \rrbracket_{d,d'}$ via $\mathcal{F}_{\text{Input}}$ (recall that each party only has one input x_i , but we can still define such a packed sharing with the other secrets being dummies). To maintain the invariant that the parties have both $\llbracket \mathbf{x} \rrbracket_{d,d'}$ and $\llbracket \mathbf{r} \cdot \mathbf{x} \rrbracket_{d,d'}$ for every batch of wires, the parties start by invoking π_{Mult} on $\llbracket \mathbf{r} \rrbracket_{d,d'}$ and $\llbracket \mathbf{x}_i \rrbracket_{d,d'}$ for each input x_i . The parties then execute the circuit, layer-by-layer, first using π_{Network} to collect the sharings for that layer, then using π_{Mult} to execute the multiplications and doing local additions to compute the gates of that layer. Note we also perform these operations on the randomized sharings $\llbracket \mathbf{r} \cdot \mathbf{x} \rrbracket_{d,d'}$ to maintain the invariant. Once the parties have sharings (and randomized sharings) of the outputs, they perform degree verification on all sharings distributed by P_1 during $\pi_{\text{SharingTrans}}$, run $\pi_{\text{CheckMult}}$ to ensure all multiplications were performed correctly, and invoke $\mathcal{F}_{\text{Open}}$ to reconstruct outputs. We present Π_{Main} below.

Protocol 8: Π_{Main}

1. Parties invoke $\mathcal{F}_{\text{CSamp}}$ to sample random committees $\mathcal{C}_1, \dots, \mathcal{C}_m$. Each input party learns all of the committees; every other party only learns which committees they are a part of, and parties in those committees.
2. Parties invoke $\mathcal{F}_{\text{RandVerif}}$ to obtain random sharings $\llbracket \mathbf{r} \rrbracket_{d,d'}$ and $\llbracket \mathbf{s} \rrbracket_{d,d'}$ (recall parties erase any state used during an invocation of $\mathcal{F}_{\text{RandVerif}}$).^a
3. Parties invoke $\mathcal{F}_{\text{DV}}$ on input **prepare**. They also invoke $\mathcal{F}_{\text{RG}}$ on input **prepare** to store random element χ (parties do not receive it yet).
4. Let p_i be the position of input party P_i 's input and $\text{pos}_i = (p_i, \gamma_2, \dots, \gamma_\rho)$, where each γ_j is some distinct unused position. Each input party invokes $\mathcal{F}_{\text{Input}}$ on their input x_i so that the parties obtain shares $\llbracket \mathbf{x}_i \rrbracket_{d,d'}$. Then they invoke π_{Mult} on $\llbracket \mathbf{x}_i \rrbracket_{d,d'}$ and $\llbracket \mathbf{r} \rrbracket_{d,d'}$ to obtain $\llbracket \mathbf{r} \cdot \mathbf{x}_i \rrbracket_{d,d'}$. Finally, the parties invoke $\pi_{\text{SharingTrans}}$ on $\llbracket \mathbf{x}_i \rrbracket_{d,d'}$ with (β, pos_i, I) , where I is the identity map to obtain $\llbracket \mathbf{x}_i \rrbracket_{d,d'}$ and $\llbracket \mathbf{r} \cdot \mathbf{x}_i \rrbracket_{d,d'}$.
5. For each circuit layer:
 - (a) The gates are divided into groups of size ρ based on their types (i.e., multiplication or addition). For each group of ρ gates, let $\mathbf{x}$ be the first inputs of all gates and $\mathbf{y}$ the second inputs of all gates. Parties invoke π_{Network} to prepare $\llbracket \mathbf{x} \rrbracket_{d,d'}$, $\llbracket \mathbf{r} \cdot \mathbf{x} \rrbracket_{d,d'}$, $\llbracket \mathbf{y} \rrbracket_{d,d'}$, $\llbracket \mathbf{r} \cdot \mathbf{y} \rrbracket_{d,d'}$,

- (b) For each group of ρ gates, let $\mathbf{pos}$ be the positions associated with the outputs of these ρ gates.
 - If they are addition gates, the parties locally compute $\llbracket \mathbf{z} \rrbracket_{d,d'} \leftarrow \llbracket \mathbf{x} \rrbracket_{d,d'} + \llbracket \mathbf{y} \rrbracket_{d,d'}$ and then invoke $\pi_{\text{SharingTrans}}$ on $\llbracket \mathbf{z} \rrbracket_{d,d'}$ with $(\beta, \mathbf{pos}, I)$, where I is the identity map, to obtain $\llbracket \mathbf{z} || \mathbf{pos} \rrbracket_{d,d'}$. They do the same with $\llbracket \mathbf{r} \cdot \mathbf{x} \rrbracket_{d,d'}$ and $\llbracket \mathbf{r} \cdot \mathbf{y} \rrbracket_{d,d'}$ to obtain $\llbracket \mathbf{r} \cdot \mathbf{z} || \mathbf{pos} \rrbracket_{d,d'}$.
 - If they are multiplication gates, the parties invoke π_{Mult} on $\llbracket \mathbf{x} \rrbracket_{d,d'}, \llbracket \mathbf{y} \rrbracket_{d,d'}$ with $\mathbf{pos}$ to obtain $\llbracket \mathbf{z} || \mathbf{pos} \rrbracket_{d,d'}$. They do the same with $\llbracket \mathbf{r} \cdot \mathbf{x} \rrbracket_{d,d'}, \llbracket \mathbf{y} \rrbracket_{d,d'}$ to obtain $\llbracket \mathbf{r} \cdot \mathbf{z} || \mathbf{pos} \rrbracket_{d,d'}$.
- 6. Output phase:
 - (a) Parties invoke $\mathcal{F}_{\text{RG}}$ on input open to open random element χ .
 - (b) Parties invoke $\mathcal{F}_{\text{DV}}$ on input verify to verify the degree of every sharing, $\llbracket \mathbf{f}(\mathbf{e}) || \mathbf{pos}' \rrbracket_{\rho-1,0}$, party P_1 sent to the rest in procedure $\pi_{\text{SharingTrans}}$ throughout the execution.
 - (c) Parties invoke $\pi_{\text{CheckMult}}$ on the regular and randomized outputs of every multiplication gate in the circuit (computed in π_{Mult}), $\llbracket \mathbf{z} \rrbracket_{d,d'}, \llbracket \mathbf{r} \cdot \mathbf{z} \rrbracket_{d,d'}$, and the randomization of the input gates, $\llbracket \mathbf{x}_i \rrbracket_{d,d'}, \llbracket \mathbf{r} \cdot \mathbf{x}_i \rrbracket_{d,d'}$, with random sharings $\llbracket \mathbf{r} \rrbracket_{d,d'}, \llbracket \mathbf{s} \rrbracket_{d,d'}$ and random coin χ .
 - (d) For each output sharing $\llbracket \mathbf{z} \rrbracket_{d,d'}$ (computed in π_{Mult}), the parties invoke $\mathcal{F}_{\text{Open}}$ to obtain and output $\mathbf{z}$.

^a for the former, only the first secret value is considered.

We present our main theorem that obtains sub-quadratic communication in the number of parties.

Theorem 5. *Protocol Π_{Main} UC-realizes $\mathcal{F}_{\text{MPC}}$ in the $(\mathcal{F}_{\text{CSamp}}, \mathcal{F}_{\text{RandVerif}}, \mathcal{F}_{\text{DV}}, \mathcal{F}_{\text{RG}}, \mathcal{F}_{\text{MapRand}})$ -hybrid model with total communication of $\tilde{O}(|I|(n + \kappa) + n^{1.5})$ bits and $\tilde{O}(n^{1.5} + |I|(n + \sqrt{n} \cdot |C|) + n \cdot (\text{depth}(C) + \sigma))$ elements of $\mathbb{F}$, where σ is the number of output wires.*

For the communication cost of our protocol, we can sum up the cost of the individual sub-protocols. We need $\tilde{O}(|I|(n + \kappa) + n^{1.5})$ bits for $\mathcal{F}_{\text{CSamp}}$, $\tilde{O}(n^{1.5})$ elements for $\mathcal{F}_{\text{DV}}$, $\mathcal{F}_{\text{RG}}$, and $\pi_{\text{CheckMult}}$, $O(n \cdot |I|)$ elements for $\mathcal{F}_{\text{RandVerif}}$, $O(\sqrt{n} \cdot |I|)$ amortized across the entire circuit C and $O(n)$ for each layer of the circuit for π_{Mult} , $\pi_{\text{SharingTrans}}$, and π_{Network} , and $\tilde{O}(n)$ elements amortized per output gate from $\mathcal{F}_{\text{Open}}$. Altogether, we get that the cost is $\tilde{O}(|I|(n + \kappa) + n^{1.5})$ bits and $\tilde{O}(n^{1.5} + |I|(n + \sqrt{n} \cdot |C|) + n \cdot (\text{depth}(C) + \sigma))$ elements of $\mathbb{F}$.

10 Supplementary Material for Lower Bounds

In this section, we show that sub-quadratic communication is impossible against more powerful adaptive rushing corruptions that may adaptively corrupt after

observing messages from honest parties in a communication round, but before the messages have been sent, and then change all the messages of these honest parties (as previously in [ACD⁺19] for Byzantine agreement). We will present a $\Omega(n^2)$ communication lower bound for MPC secure with aborts against these stronger rushing adversaries for corruption threshold $t = \Omega(n)$.

Theorem 6. *Consider any MPC protocol that supports functionalities for two input parties $|I| = 2$ and every party receives the output. If the protocol is secure with abort against adversaries with strong adaptive rushing corruptions that corrupt up to t parties, then the MPC protocol requires $tn/4$ communication even for circuits with a single multiplication gate with size $|C| = 1$.*

In other words, sub-quadratic communication MPC is impossible whenever we consider these stronger adversaries that may corrupt $t = \Omega(n)$ parties. We also show our techniques may be extended to other settings including MPC protocols with deterministic (or predictable) communication patterns even against standard adaptive rushing adversaries.

To start, we prove the following lemma that considers adversaries that are able to isolate a single party P such that all parties that send a message to party P or receive a message from party P are corrupted by the adversary. We show that, if such an adversary exists, we can guarantee that the MPC protocol cannot be secure with aborts.

Lemma 8. *Consider any MPC protocol that supports functionalities for two input parties $|I| = 2$ and every party receives the output. If there exists an adversary that is able to identify a single party P and corrupt all the parties before they communicate with party P with non-negligible probability, then the MPC protocol cannot be secure with abort.*

Proof. We start with the case that the corrupted parties consist of at most one (or none) of the $|I| = 2$ input parties. In this case, the corrupted parties can simulate the protocol assuming the output bit is always 0. Since the chosen party P only interacts with the corrupted parties, the view of party P will ensure that it always outputs 0. This is the case even when the two input bits are 1 and the output bit should be 1, meaning this scheme does not satisfy security with abort.

Next, we consider the other case when both input parties are corrupted. In this case, the adversary sends messages in the following manner. For all messages to the chosen party P , the corrupted parties will simulate the protocol as if the output was 0. Since the chosen party P only interacts with these corrupted parties, its output must be 0. In contrast, the adversary will simulate the protocol as if the output is 1 when sending messages to all other honest parties except party P . As a result, this contradicts security with abort as honest party P will output 0, which is different than the 1-bit output of all other honest parties in the protocol execution. $\square$

In the remaining portion of this section, we will show when it is possible to construct such an adversary compatible with Lemma 8 that rules out the

existence of MPC with certain communication efficiency (including those with sub-quadratic communication).

Adversaries with Stronger Adaptive Corruptions. First, we prove Theorem 6 showing MPC protocols secure against adversaries with stronger adaptive corruption capabilities required $\Omega(n^2)$ communication. First, we define strong adaptive rushing corruption as follows:

1. All honest parties announce their messages for the round.
2. With this information, the adversary may choose to corrupt any subset of honest parties as long as the adversary never corrupts more than t parties.
3. The adversary will erase the messages that were to be sent by the newly corrupted parties and replace them with messages of their choice.

Proof of Theorem 6. We will consider the functionality of computing the AND function with two input parties such that $|I| = 2$ and $|C| = 1$. Towards a contradiction, suppose that there exists such a MPC protocol for this functionality with less than $tn/4$ communication. We will construct an adversary that satisfies the conditions of Lemma 8 of identifying a party P and corrupting all parties that will send messages to P in the protocol's execution with non-negligible probability.

The adversary first picks a single party P uniformly at random. In expectation, the party P will receive and send at most $(tn/4) \cdot 2/n = t/2$ messages in total. Note, we multiply by two since each message is sent and received. By applying Markov's inequality, the randomly chosen party P receives and sends at most t messages throughout the entire protocol with probability at least $1/2$. This also means that the chosen party communicates with at most t other parties. These will be the t parties that the adversary corrupts. In each communication round, the adversary checks if any honest parties will send messages to the chosen party P or receive any messages from the chosen party P . If so, the adversary corrupts the honest party to control the messages being sent to party P , and to ensure that all messages sent by P are received only by corrupted parties. In the end, all messages received by party P are controlled by the adversary and all sent messages by party P are received by the adversary with probability at least $1/2$. By Lemma 8, we immediately obtain that the MPC protocol is not secure against aborts. $\square$

We note our lower bounds are similar in spirit to prior works such as Blum *et al.* [BBCLZ23], but use different underlying problem requirements. In our proof, we rely on the security with abort guarantee as well as adversaries with strong adaptive corruption capabilities whereas the proof in [BBCLZ23] relies upon guaranteed output delivery but considers the standard adversarial model.

Deterministic Communication Patterns. We additionally consider the case that the MPC protocol results in a deterministic communication pattern. In particular, we will consider the case that whether one party sends a message to another party in a communication round does not depend on the internal randomness in any execution of the MPC protocol. In this case, we show that an

MPC protocol with less than tn communication enables an adversary to isolate such a party P as required in Lemma 8. Note, this formalizes the ideas presented in Section 2 about barriers that exist for existing state-of-the-art MPC protocols including [DIK10, GPS22, EGP⁺23, BY25, SY25].

Theorem 7. *Consider any MPC protocol that supports functionalities for two input parties $|I| = 2$ and every party receives the output with a deterministic communication pattern. If the protocol is secure with abort in the presence of a malicious adversary that corrupts up to t parties, then the MPC protocol requires $(t + 1)n/2$ communication even for circuits with a single multiplication gate with size $|C| = 1$.*

Proof. Towards a contradiction, suppose that the MPC protocol has strictly less than $(t + 1)n/2$ communication. Then, there must exist a single party P that sends and receives at most t messages in the entire protocol. In particular, this implies that party P sends and receives messages from at most t other parties. Since the communication pattern is deterministic, the adversary can corrupt the at most t parties communicating with party P . Afterwards, we can apply Lemma 8 to show that the MPC protocol is not secure with aborts. $\square$

Once again, this rules out sub-quadratic communication MPC protocols where $t = \Omega(n)$ parties may be corrupted. It also explains why most prior works [DIK10, GPS22, EGP⁺23, BY25, SY25] require an additive $\Omega(n^2)$ term in their communication costs. In general, the above attack may be extended to any MPC protocol where the adversary may predict communication patterns such that it can identify a single party that receives messages from at most t other distinct parties, which happens to be the case in the setting of deterministic communication patterns. In particular, this can also be extended to MPC protocols with randomized communication patterns where it is still possible for an adversary to find a party P and corrupt all parties communicating with P with non-negligible probability.

Acknowledgements. Part of this work was done while the last author was at Columbia University that was partially supported by NSF grant CCF-2312242.

Disclaimer. This paper was prepared for informational purposes in part by the Artificial Intelligence Research group of JPMorgan Chase & Co. and its affiliates ("JP Morgan") and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

- ACD⁺19. Ittai Abraham, TH Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 317–326, 2019.
- AJL⁺12. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Berlin, Heidelberg, April 2012.
- ALM17. Adi Akavia, Rio LaVigne, and Tal Moran. Topology-hiding computation on all graphs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 447–467. Springer, Cham, August 2017.
- AM17. Adi Akavia and Tal Moran. Topology-hiding computation beyond logarithmic diameter. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 609–637. Springer, Cham, April / May 2017.
- BBCLZ23. Erica Blum, Elette Boyle, Ran Cohen, and Chen-Da Liu-Zhang. Communication lower bounds for cryptographic broadcast protocols. In *37th International Symposium on Distributed Computing*, 2023.
- BBK⁺24. James Bartusek, Thiago Bergamaschi, Seri Khoury, Saachi Mutreja, and Orr Paradise. On the communication complexity of secure multi-party computation with aborts. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *43rd ACM PODC*, pages 480–491. ACM, June 2024.
- BBKM23. Marshall Ball, Alexander Bienstock, Lisa Kohl, and Pierre Meyer. Towards topology-hiding computation from oblivious transfer. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part I*, volume 14369 of *LNCS*, pages 349–379. Springer, Cham, November / December 2023.
- BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Berlin, Heidelberg, May 2011.
- BEO17. Aner Ben-Efraim and Eran Omri. Concrete efficiency improvements for multiparty garbling with an honest majority. In *International Conference on Cryptology and Information Security in Latin America*, pages 289–308. Springer, 2017.
- BEP23. Alexander Bienstock, Daniel Escudero, and Antigoni Polychroniadou. On linear communication complexity for (maximally) fluid MPC. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 263–294. Springer, Cham, August 2023.
- BEP24. Alexander Bienstock, Daniel Escudero, and Antigoni Polychroniadou. Perfectly secure fluid MPC with abort and linear communication complexity. *CiC*, 1(4):28, 2024.
- BGJK21. Gabrielle Beck, Aarushi Goel, Abhishek Jain, and Gabriel Kaptchuk. Order-C secure multiparty computation for highly repetitive circuits. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 663–693. Springer, Cham, October 2021.

- BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.
- BH93. Donald Beaver and Stuart Haber. Cryptographic protocols provably secure against dynamic adversaries. In Rainer A. Rueppel, editor, *EUROCRYPT’92*, volume 658 of *LNCS*, pages 307–323. Springer, Berlin, Heidelberg, May 1993.
- BHK⁺22. Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Alex Miao, and Tal Rabin. Threshold cryptography as a service (in the multiserver and YOSO models). In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 323–336. ACM Press, November 2022.
- Bra87. Gabriel Bracha. An $o(\log n)$ expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, October 1987.
- BY25. Alexander Bienstock and Kevin Yeo. Weakly super-invertible matrices and constant communication dishonest majority MPC. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part V*, volume 15605 of *LNCS*, pages 421–452. Springer, Cham, May 2025.
- Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
- CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.
- CDdH07. Ronald Cramer, Ivan Damgård, and Robbert de Haan. Atomic secure multi-party multiplication with low communication. In *EUROCRYPT 2007*, volume 4515, pages 329–346. Springer, 2007.
- CDGK22. Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 651–680. Springer, Cham, December 2022.
- CGG⁺21. Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: Secure multiparty computation with dynamic participants. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 94–123, Virtual Event, August 2021. Springer, Cham.
- CGH⁺18. Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 34–64. Springer, Cham, August 2018.
- CLMZ23. Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. Eos: Efficient private delegation of zkSNARK provers. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 6453–6469. USENIX Association, August 2023.
- CsW19. Ran Cohen, abhi shelat, and Daniel Wichs. Adaptively secure MPC with sublinear communication complexity. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 30–60. Springer, Cham, August 2019.

- CSW23. Ran Cohen, Abhi Shelat, and Daniel Wichs. Adaptively secure mpc with sublinear communication complexity. *J. Cryptol.*, 36(2), March 2023.
- DDG⁺23. Bernardo David, Giovanni Deligios, Aarushi Goel, Yuval Ishai, Anders Konring, Eyal Kushilevitz, Chen-Da Liu-Zhang, and Varun Narayanan. Perfect MPC over layered graphs. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 360–392. Springer, Cham, August 2023.
- DIK⁺08. Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 241–261. Springer, Berlin, Heidelberg, August 2008.
- DIK10. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 445–465. Springer, Berlin, Heidelberg, May / June 2010.
- DLN19. Ivan Damgård, Kasper Green Larsen, and Jesper Buus Nielsen. Communication lower bounds for statistically secure MPC, with or without preprocessing. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 61–84. Springer, Cham, August 2019.
- DLS21. Ivan Bjerre Damgård, Boyang Li, and Nikolaj Ignatieff Schwartzbach. More communication lower bounds for information-theoretic MPC. In Stefano Tessaro, editor, *ITC 2021*, volume 199 of *LIPIcs*, pages 2:1–2:18. Schloss Dagstuhl, July 2021.
- DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 572–590. Springer, Berlin, Heidelberg, August 2007.
- DNOR16. Ivan Damgård, Jesper Buus Nielsen, Rafail Ostrovsky, and Adi Rosén. Unconditionally secure computation with reduced interaction. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 420–447. Springer, Berlin, Heidelberg, May 2016.
- DNPR16. Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, and Michael Raskin. On the communication required for unconditionally secure multiplication. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 459–488. Springer, Berlin, Heidelberg, August 2016.
- DPP14. Deepesh Data, Manoj Prabhakaran, and Vinod M. Prabhakaran. On the communication complexity of secure computation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 199–216. Springer, Berlin, Heidelberg, August 2014.
- DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Berlin, Heidelberg, August 2012.
- EGP⁺23. Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, Yifan Song, and Chenkai Weng. SuperPack: Dishonest majority MPC with constant online communication. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 220–250. Springer, Cham, April 2023.

- EMP25. Daniel Escudero, Elisaweta Masserova, and Antigoni Polychroniadou. Towards scalable YOSO MPC via packed secret-sharing. *Cryptology ePrint Archive*, Paper 2025/635, 2025.
- FGR⁺25. Zhiyong Fang, Sanjam Garg, Bhaskar Roberts, Wenxuan Wu, and Yupeng Zhang. Collaborative zkSNARKs with sublinear prover time and constant proof size. *Cryptology ePrint Archive*, Paper 2025/1388, 2025.
- FT25. Hanwen Feng and Qiang Tang. Asymptotically optimal adaptive asynchronous common coin and DKG with silent setup. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 642–675. Springer, Cham, August 2025.
- FY92. Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710. ACM Press, May 1992.
- GGJ⁺23. Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-knowledge SNARKs as a service. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 4427–4444. USENIX Association, August 2023.
- GGJ⁺25. Sanjam Garg, Aarushi Goel, Abhishek Jain, Bhaskar Roberts, and Sruthi Sekar. Malicious security in collaborative zk-SNARKs: More than meets the eye. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 396–428. Springer, Cham, August 2025.
- GHK⁺21. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Cham.
- GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 465–482. Springer, Berlin, Heidelberg, April 2012.
- GIOZ17. Juan A. Garay, Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. The price of low communication in secure multi-party computation. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 420–446. Springer, Cham, August 2017.
- GIP⁺14. Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In David B. Shmoys, editor, *46th ACM STOC*, pages 495–504. ACM Press, May / June 2014.
- GIP15. Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. Efficient multi-party computation: From passive to active security via secure SIMD circuits. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 721–741. Springer, Berlin, Heidelberg, August 2015.
- GLO⁺21. Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. ATLAS: Efficient and scalable MPC in the honest majority setting. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 244–274, Virtual Event, August 2021. Springer, Cham.

- 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.
- GPS21. Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Unconditional communication-efficient MPC via hall’s marriage theorem. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 275–304, Virtual Event, August 2021. Springer, Cham.
- GPS22. Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 3–32. Springer, Cham, August 2022.
- GSY21. S. Dov Gordon, Daniel Starin, and Arkady Yerukhimovich. The more the merrier: Reducing the cost of large scale MPC. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 694–723. Springer, Cham, October 2021.
- HIK07. Danny Harnik, Yuval Ishai, and Eyal Kushilevitz. How many oblivious transfers are needed for secure multiparty computation? In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 284–302. Springer, Berlin, Heidelberg, August 2007.
- HLP11. Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 132–150. Springer, Berlin, Heidelberg, August 2011.
- HLP15. Carmit Hazay, Yehuda Lindell, and Arpita Patra. Adaptively secure computation with partial erasures. In Chryssis Georgiou and Paul G. Spirakis, editors, *34th ACM PODC*, pages 291–300. ACM, July 2015.
- HMTZ16. Martin Hirt, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Network-hiding communication and applications to multi-party protocols. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 335–365. Springer, Berlin, Heidelberg, August 2016.
- HMW⁺25. Yuncong Hu, Pratyush Mishra, Xiao Wang, Jie Xie, Kang Yang, Yu Yu, and Yuwen Zhang. Dfs: Delegation-friendly zkSNARK and private delegation of provers. In *USENIX Security 2025*, 2025.
- HOSS18a. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Concretely efficient large-scale MPC with active security (or, TinyKeys for TinyOT). In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 86–117. Springer, Cham, December 2018.
- HOSS18b. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. TinyKeys: A new approach to efficient multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 3–33. Springer, Cham, August 2018.
- IPS09. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 294–314. Springer, Berlin, Heidelberg, March 2009.
- KKK⁺10. Bruce M Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. *ACM Transactions on Algorithms (TALG)*, 6(4):1–28, 2010.

- KMR11. Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011.
- KMR12. Seny Kamara, Payman Mohassel, and Ben Riva. Salus: a system for server-aided secure function evaluation. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 797–808. ACM Press, October 2012.
- KS11. Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. *Journal of the ACM (JACM)*, 58(4):1–24, 2011.
- KTZ13. Jonathan Katz, Aishwarya Thiruvengadam, and Hong-Sheng Zhou. Feasibility and infeasibility of adaptively secure fully homomorphic encryption. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 14–31. Springer, Berlin, Heidelberg, February / March 2013.
- Lin09. Andrew Y. Lindell. Adaptively secure two-party computation with erasures. In Marc Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, pages 117–132, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- LSP19. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the works of leslie lamport*, pages 203–226, 2019.
- LZW⁺24. Xuanming Liu, Zhelei Zhou, Yinghao Wang, Bingsheng Zhang, and Xiaohu Yang. Scalable collaborative zk-SNARK: Fully distributed proof generation and malicious security. Cryptology ePrint Archive, Report 2024/143, 2024.
- LZW⁺25. Xuanming Liu, Shelei Zhou, Yinghao Wang, Yanxin Pang, Jinye He, Bingsheng Zhang, Xiaohu Yang, and Jiaheng Zhang. Scalable collaborative zk-snark and its application to fully distributed proof delegation. In *USENIX Security 2025*, 2025.
- MOR15. Tal Moran, Ilan Orlov, and Silas Richelson. Topology-hiding computation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 159–181. Springer, Berlin, Heidelberg, March 2015.
- NNOB12. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Berlin, Heidelberg, August 2012.
- NR17. Jesper Buus Nielsen and Samuel Ranellucci. On the computational overhead of MPC with dishonest majority. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 369–395. Springer, Berlin, Heidelberg, March 2017.
- OB22. Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 4291–4308. USENIX Association, August 2022.
- RS22. Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 719–749. Springer, Cham, August 2022.
- Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
- SV25. Alessandra Scafuro and Tanner Verber. A new paradigm for server-aided mpc. *IACR Communications in Cryptology*, 1(4), 2025.

- SY25. Yifan Song and Xiaxi Ye. Honest majority MPC with $\tilde{O}(|C|)$ communication in minicrypt. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part V*, volume 15605 of *LNCS*, pages 453–483. Springer, Cham, May 2025.
- Yao82. Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.
- YCW⁺25. Yunbo Yang, Yuejia Cheng, Kailun Wang, Xiaoguo Li, Jianfei Sun, Jiachen Shen, Xiaolei Dong, Zhenfu Cao, Guomin Yang, and Robert H. Deng. Siniel: Distributed privacy-preserving zkSNARK. In *NDSS 2025*. The Internet Society, February 2025.

A Universal Composability [Can01]

We work in the Universal-Composability (UC) framework of Canetti [Can01]. In this framework, for a given protocol Π , executed by parties, and a functionality $\mathcal{F}$ we have an environment $\mathcal{Z}$ and an adversary $\mathcal{A}$, and we consider two worlds.

1. **Real world execution.** In the real world execution, the adversary $\mathcal{A}$ interacts with n parties and can adaptively corrupt up to t of them (see note on adaptive corruption in Section 4). At the end of the protocol, the adversary $\mathcal{A}$ sends $\mathcal{Z}$ its entire view from the protocol, i.e. the messages it received and sent in each round, and the internal states of all corrupt parties. $\mathcal{Z}$ also receives the outputs of all honest parties. Denote the distribution of views seen by $\mathcal{Z}$ with adversary $\mathcal{A}$ in the real world execution by $\mathbf{D}_{\mathcal{R},\mathcal{A}}$
2. **Ideal world execution.** In the ideal world execution, the adversary $\mathcal{A}$ interacts with a simulator $\mathcal{S}$, that simulates the n parties, and allows the adversary to adaptively corrupt at most t of them. Furthermore, $\mathcal{S}$ has black-box access to $\mathcal{F}$. At the end of the interaction, the adversary $\mathcal{A}$ sends to $\mathcal{Z}$ its entire view from the protocol, and $\mathcal{F}$ sends to $\mathcal{Z}$ the outputs of the honest parties. Denote the distribution of views seen by $\mathcal{Z}$ with adversary $\mathcal{A}$ and simulator $\mathcal{S}$ in the ideal world execution by $\mathbf{D}_{\mathcal{S},\mathcal{A}}$

We say that a protocol Π UC-realizes a functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ s.t. $\mathbf{D}_{\mathcal{R},\mathcal{A}} \stackrel{\epsilon}{\approx} \mathbf{D}_{\mathcal{S},\mathcal{A}}$, for $\epsilon = \text{negl}(\kappa)$, where κ is the security parameter. In this work, we focus on statistical security, so $\mathcal{A}$ can run in unbounded time. Sometimes, we will also utilize the hybrid model of UC. In this case, in the real world the honest parties have access to some set of sub-functionalities $\{\mathcal{F}_i\}_{i \in [m]}$, and we denote the distribution of views in this case as $\mathbf{D}_{\mathcal{R},\mathcal{A}}^{\{\mathcal{F}_i\}_{i \in [m]}}$. In the ideal world, the simulator emulates these sub-functionalities, and we denote the distribution of views in this case as $\mathbf{D}_{\mathcal{S},\mathcal{A}}^{\{\mathcal{F}_i\}_{i \in [m]}}$. We say that a protocol Π UC-realizes a functionality $\mathcal{F}$ in the $(\{\mathcal{F}_i\}_{i \in [m]})$ -hybrid model if for any adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ s.t. $\mathbf{D}_{\mathcal{R},\mathcal{A}}^{\{\mathcal{F}_i\}_{i \in [m]}} \stackrel{\epsilon}{\approx} \mathbf{D}_{\mathcal{S},\mathcal{A}}^{\{\mathcal{F}_i\}_{i \in [m]}}$, for $\epsilon = \text{negl}(\kappa)$.

B Additional MPC Building Blocks

In this section, we will instantiate the remaining building blocks common to statistically-secure MPC Protocol.

B.1 Randomness Generation

Here we introduce the functionality $\mathcal{F}_{\text{RG}}$ for generating a random element $\chi \in \mathbb{F}$. Randomness generation has both a prepare and open phase. The former is used to sample (but not reveal) the random value χ , while the latter actually opens it to all parties.

Functionality 5: $\mathcal{F}_{\text{RG}}$

On input prepare: $\mathcal{F}_{\text{RG}}$ waits for an input from the adversary. If it is **abort**, $\mathcal{F}_{\text{DV}}$ outputs **abort** to all parties; otherwise, it samples random $\chi \in \mathbb{F}$.
On input open: $\mathcal{F}_{\text{RG}}$ waits for an input from the adversary. If it is **abort**, $\mathcal{F}_{\text{DV}}$ outputs **abort** to all parties; otherwise, it outputs χ to all parties.

$\mathcal{F}_{\text{RG}}$ is instantiated by the below protocol Π_{RG} , using standard techniques based on π_{Rand} .

Protocol 9: Π_{RG}

On input prepare: The parties run $\pi_{\text{Rand}}(1, d, d')$ to obtain random sharing $\llbracket \chi \rrbracket_{d,d'}$. **On input open:** The parties invoke $\mathcal{F}_{\text{Open}}$ on $\llbracket \chi \rrbracket_{d,d'}$ to open χ and output the first element χ_1 .

The communication complexity of Π_{RG} is $O(n|I|)$ elements for π_{Rand} and $\tilde{O}(n^{1.5})$ elements for $\mathcal{F}_{\text{Open}}$. $\mathcal{F}_{\text{RG}}$ will only be invoked a constant number of times throughout the entire main MPC protocol, Π_{Main} .

Lemma 9. Π_{RG} UC-realizes $\mathcal{F}_{\text{RG}}$ in the $\mathcal{F}_{\text{Open}}$ -hybrid model.

Proof. This proof follows easily. Indeed, the simulator simulates π_{Rand} by just sampling and distributing random sharing $\llbracket r_{i,j} \rrbracket_{d,d'}$ on behalf of all honest parties. $\mathcal{F}_{\text{Open}}$ aborts if $\llbracket \chi \rrbracket_{d,d'}$ is not of the correct degree, and thus if there is at least one honest party, the distribution of the output χ_1 in the real world is random and identically distributed to the ideal world. $\square$

B.2 Degree Verification

In this section, we present the functionality and protocol for degree-verification of a batch of virtual sharings $\llbracket x_1 \rrbracket_{d,d'}, \dots, \llbracket x_k \rrbracket_{d,d'}$. If any such sharing is of the wrong degree, the honest parties should abort. Degree verification has both a prepare and verification phase. The former is used to sample (but not reveal) challenge randomness, while the latter reveals that challenge randomness to perform the verification. First, we present the functionality $\mathcal{F}_{\text{DV}}$.

Functionality 6: $\mathcal{F}_{\text{DV}}$

On input prepare: $\mathcal{F}_{\text{DV}}$ waits for an input from the adversary. If it is abort, $\mathcal{F}_{\text{DV}}$ outputs abort to all parties; otherwise, it does nothing.

On input verify($\llbracket \mathbf{x}_1 \rrbracket_{d,d'}, \dots, \llbracket \mathbf{x}_k \rrbracket_{d,d'} \rrbracket$):

1. $\mathcal{F}_{\text{DV}}$ receives from the honest parties their shares.
2. $\mathcal{F}_{\text{DV}}$ recovers the whole inner sharings $[x_j^k]_{d'}$ for those committees $\mathcal{C}_k$ in which $|\mathcal{H}_k| > d'$ and sends the shares of the corrupted parties in these committees to the adversary. Moreover, based on the above, $\mathcal{F}_{\text{DV}}$ recovers the whole outer sharings $[\mathbf{x}_j]_d$, including the secrets $\mathbf{x}_j$.
3. For each committee $\mathcal{C}_k$ in which $|\mathcal{T}_k| > d'$, $\mathcal{F}_{\text{DV}}$ sends to the adversary the honest parties' shares $\{\mathbf{x}_j^{k,h}\}_{h \in \mathcal{H}_i}$ of the corresponding inner sharing, and the corresponding outer shares $\mathbf{x}_j^k$.
4. If any of the inner sharings for those committees $\mathcal{C}_k$ in which $|\mathcal{H}_k| > d'$ are not of degree d' , or any of the outer sharings are not of degree d , $\mathcal{F}_{\text{DV}}$ sends abort to the parties.

Note that sometimes, the parties will invoke $\mathcal{F}_{\text{DV}}$ with **verify** immediately after invoking it with **prepare**; in this case, we will just say that the parties invoke $\mathcal{F}_{\text{DV}}$.

Now we present the protocol Π_{DV} . This protocol uses the standard random linear combination technique.

Protocol 10: Π_{DV}

On input prepare: The parties invoke $\mathcal{F}_{\text{RG}}$ on input **prepare** to store random element χ (the parties do not receive it yet).

On input verify($\llbracket \mathbf{x}_1 \rrbracket_{d,d'}, \dots, \llbracket \mathbf{x}_\ell \rrbracket_{d,d'} \rrbracket$):

1. Each input party P_i samples random vector $\mathbf{r}_i$ and random virtual sharing $\llbracket \mathbf{r}_i \rrbracket_{d,d'}$ and distributes the shares to the other parties.
2. The parties compute $\llbracket \mathbf{r} \rrbracket_{d,d'} \leftarrow \sum_{i \in I} \llbracket \mathbf{r}_i \rrbracket_{d,d'}$.
3. The parties invoke $\mathcal{F}_{\text{RG}}$ on input **open** to obtain random χ .
4. The parties compute $\llbracket \mathbf{e} \rrbracket_{d,d'} \leftarrow \sum_{j=1}^{\ell} \chi^j \cdot \llbracket \mathbf{x}_j \rrbracket_{d,d'} + \llbracket \mathbf{r} \rrbracket_{d,d'}$.
5. The parties invoke $\mathcal{F}_{\text{Open}}$ to open $\llbracket \mathbf{e} \rrbracket_{d,d'}$ to all parties.

The communication complexity of Π_{DV} is $O(n|I|)$ elements for the distribution of $\llbracket \mathbf{r}_i \rrbracket_{d,d'}$ and $\tilde{O}(n^{1.5})$ elements for $\mathcal{F}_{\text{RG}}$ and $\mathcal{F}_{\text{Open}}$. $\mathcal{F}_{\text{DV}}$ is invoked a constant number of times throughout the entire main MPC protocol, Π_{Main} .

Lemma 10. Π_{DV} UC-realizes $\mathcal{F}_{\text{DV}}$ in the $\mathcal{F}_{\text{RG}}$ -hybrid model.

Proof. We begin with a simulator $\mathcal{S}$, which initializes **should-abort** $\leftarrow 0$:

1. $\mathcal{S}$ first receives from $\mathcal{F}_{\text{DV}}$, for those committees $\mathcal{C}_k$ in which $|\mathcal{H}_k| > d'$, the inner shares of $\llbracket \mathbf{x}_1 \rrbracket_{d,d'}, \dots, \llbracket \mathbf{x}_\ell \rrbracket_{d,d'}$ of the corrupted parties in these

- committees. It also receives the outer shares of those committees in which $|\mathcal{T}_k| > d'$.
2. Next for each honest input party, $\mathcal{S}$ samples $\llbracket \mathbf{r}_i \rrbracket_{d,d'}$ and sends the corrupted parties' shares to the adversary.
 3. $\mathcal{S}$ receives the same from the adversary for the corrupted input parties and reconstructs the corresponding $\llbracket \mathbf{r}_i \rrbracket_{d,d'}$ (non-corrupted inner sharings and whole outer sharing).
 4. $\mathcal{S}$ then computes $\llbracket \mathbf{r} \rrbracket_{d,d'}$. If it is of the wrong degree, $\mathcal{S}$ sets **should-abort** $\leftarrow 1$.
 5. $\mathcal{S}$ emulates $\mathcal{F}_{\text{RG}}$ by sampling random χ and sending it to the adversary.
 6. $\mathcal{S}$ computes the corrupted parties' shares of $\llbracket \mathbf{e} \rrbracket_{d,d'}$ (except for corrupted inner sharings).
 7. Next, $\mathcal{S}$ emulates $\mathcal{F}_{\text{Open}}$ by receiving from the adversary the corrupted parties' shares of $\llbracket \mathbf{e} \rrbracket_{d,d'}$ and then sending $\mathbf{r}$ to the adversary; if any of the received shares are unequal to those which $\mathcal{S}$ computed above, $\mathcal{S}$ sends **abort** to $\mathcal{F}_{\text{DV}}$.
 8. Finally if **should-abort** = 1, $\mathcal{S}$ sends **abort** to $\mathcal{F}_{\text{DV}}$.

We now show that the real and ideal worlds are indistinguishable. It is clear that $\mathcal{S}$'s sampling of $\llbracket \mathbf{r}_i \rrbracket_{d,d'}$ and emulation of $\mathcal{F}_{\text{RG}}$ is identical to the real world. Moreover, for the emulation of $\mathcal{F}_{\text{Open}}$, since each $\mathbf{r}_i$ of honest input parties are random, so too is $\mathbf{r}$ and thus so too is $\mathbf{e}$; thus this emulation is identical to the real world. Now it remains to show that the output distributions are the same. There is an abort in the ideal world if any of the $\llbracket \mathbf{x}_j \rrbracket_{d,d'}$ or $\llbracket \mathbf{r} \rrbracket_{d,d'}$ is not of the proper degree, which can happen in two ways:

1. An inner sharing $\llbracket \mathbf{x}_j^k \rrbracket_{d,d'}$ or $\llbracket \mathbf{r}^k \rrbracket_{d,d'}$ for a committee in which $|\mathcal{H}_k| > d' + 1$ is not of the right degree. Consider the polynomial $[F(\mathbf{y})]_{d'} := \llbracket \mathbf{r}^k \rrbracket_{d'} + \sum_{j=1}^{\ell} \llbracket \mathbf{x}_j^k \rrbracket \cdot \mathbf{y}^j$. By Lagrange interpolation, for any $\ell + 1$ different points $\chi_1, \dots, \chi_{\ell+1}$, there is a one-to-one linear map from $\{[F(\chi_j)]_{d'}\}_{j \in [\ell+1]}$ to $\{\llbracket \mathbf{r}^k \rrbracket\} \cup \{\llbracket \mathbf{x}_j^k \rrbracket\}_{j \in [\ell]}$. So, if either $\llbracket \mathbf{r}^k \rrbracket_{d,d'}$ or some $\llbracket \mathbf{x}_j^k \rrbracket_{d,d'}$ is not of the correct degree, then the number of $\chi \in \mathbb{F}$ such that the shares of $[F(\chi)]$ are of the correct degree is bounded by ℓ . Since χ is uniform in $\mathbb{F}$, the probability of sampling such a bad χ is bounded by $\ell/|\mathbb{F}|$, which is negligible.
2. An outer sharing is not of the right degree. This also follows the same argument as the first case; indeed, the number of committees in which $|\mathcal{H}_{k^*}| > d' + 1$, and thus the honest parties' inner shares of that committee define the corresponding share of the outer sharing, is greater than $d + 1$.

There is also an abort in the ideal world if the corrupted parties do not input the correct shares of $\llbracket \mathbf{e} \rrbracket_{d,d'}$ to $\mathcal{F}_{\text{Open}}$; such an abort would also happen in the real world, because of the error detection properties of the sharing scheme.

Thus the two worlds are indistinguishable. $\square$

B.3 Virtual Random Sharings

We now present the functionality for sampling random virtual sharings $\llbracket \mathbf{r} \rrbracket_{d,d'}$.

Functionality 7: $\mathcal{F}_{\text{RandVerif}}$

1. $\mathcal{F}_{\text{RandVerif}}$ first may receive **abort** from the adversary; if so, it outputs **abort** to the honest parties.
2. Otherwise, $\mathcal{F}_{\text{RandVerif}}$ receives from the adversary the set of outer shares $\{r^i\}_{i \in \mathcal{T}_{\text{virt}}}$ of the corrupted virtual parties.
3. Also, for each committee $\mathcal{C}_i$ in which $|\mathcal{T}_i| \leq d'$, $\mathcal{F}_{\text{RandVerif}}$ receives from the adversary the set of corrupted parties' corresponding inner shares $\{r^{ij}\}_{j \in \mathcal{T}_i}$.
4. Finally, for each committee $\mathcal{C}_i$ in which $|\mathcal{T}_i| > d'$, $\mathcal{F}_{\text{RandVerif}}$ receives the honest parties' corresponding inner share $\{r^{ij}\}_{j \in \mathcal{H}_i}$.
5. $\mathcal{F}_{\text{RandVerif}}$ samples a random $\mathbf{r}$ and random virtual sharing $\llbracket \mathbf{r} \rrbracket_{d,d'}$ consistent with the shares received above, then distributes to the honest parties their shares.

Next, we present the protocol $\Pi_{\text{RandVerif}}(\ell)$, which samples ℓ random sharings at once. We rely on the input parties to sample random sharings, which the parties sum up; if there is an honest input party, then the resulting sharing will be random. We also use $\mathcal{F}_{\text{DV}}$ to ensure that the sharings are of proper degree.

Protocol 11: $\Pi_{\text{RandVerif}}(\ell)$

1. The parties run $\pi_{\text{Rand}}(\ell, d, d')$ to obtain random sharings $(\llbracket \mathbf{r}_1 \rrbracket_{d,d'}, \dots, \llbracket \mathbf{r}_\ell \rrbracket_{d,d'})$.
2. The parties invoke $\mathcal{F}_{\text{DV}}$ on $(\llbracket \mathbf{r}_1 \rrbracket_{d,d'}, \dots, \llbracket \mathbf{r}_\ell \rrbracket_{d,d'})$; if it does not abort, they output their sharings.

The communication complexity of the above protocol for random virtual sharings $\llbracket \mathbf{r} \rrbracket_{d,d'}$ is $O(|I| \cdot n \cdot \ell)$ elements for the invocations of π_{Rand} and $\tilde{O}(n^{1.5})$ elements for $\mathcal{F}_{\text{DV}}$. $\Pi_{\text{RandVerif}}$ can be invoked once for all needed sharings at the beginning of the main MPC protocol Π_{Main} . This is $O(|I|\sqrt{n})$ amortized for each of the $\sqrt{n}$ components $\mathbf{r}_{j,i}$ within each of the ℓ output sharings of $\mathbf{r}_j$.

Lemma 11. $\Pi_{\text{RandVerif}}$ UC-realizes $\mathcal{F}_{\text{RandVerif}}$ in the $\mathcal{F}_{\text{DV}}$ -hybrid model.

Proof. We begin by constructing a simulator $\mathcal{S}$ for $\Pi_{\text{RandVerif}}$, which initializes $\text{should-abort} \leftarrow 0$:

1. On behalf of each honest input party, P_i , $\mathcal{S}$ samples random sharings $\llbracket \mathbf{r}_{i,j} \rrbracket_{d,d'}$, for $j \in [\ell]$, then sends to the adversary the corrupted parties' shares.
2. Next, $\mathcal{S}$ receives from the adversary the honest parties' shares of the corrupted input parties' sharings, $\llbracket \mathbf{r}_{i,j} \rrbracket_{d,d'}$, for $j \in [\ell]$.
3. $\mathcal{S}$ then computes the honest parties' shares of $\llbracket \mathbf{r}_j \rrbracket_{d,d'} \leftarrow \sum_{i \in I} \llbracket \mathbf{r}_{i,j} \rrbracket_{d,d'}$ for $j \in [\ell]$.
4. For each $j \in [\ell]$, and any committees in which $|\mathcal{H}_k| \geq d' + 1$:
 - (a) If the honest parties' shares of the corresponding inner sharing of $\llbracket \mathbf{r}_j \rrbracket_{d,d'}$ do not lie on a degree- d' polynomial, $\mathcal{S}$ sets $\text{should-abort} \leftarrow 1$.

- (b) Either way, $\mathcal{S}$ reconstructs the whole inner sharing (of minimal degree) based on the honest parties' shares, including the secret, which is the corresponding outer share of that committee $\mathbf{r}_j^k$.
- 5. Since the number of committees in which $|\mathcal{H}_k| \geq d' + 1$ is at least $d + 1$, $\mathcal{S}$ can check if the corresponding shares of the outer sharing of $\llbracket \mathbf{r}_j \rrbracket_{d,d'}$ lie on a degree- d polynomial.
- 6. If not, $\mathcal{S}$ sets **should-abort** $\leftarrow 1$. Either way, $\mathcal{S}$ reconstructs the whole outer sharing (of minimal degree) based on the outer shares it obtained above.
- 7. $\mathcal{S}$ emulates $\mathcal{F}_{\text{DV}}$ by first receiving from the corrupted parties their shares of the above; if for any committee such that $|\mathcal{H}_k| > d'$, the corrupted parties' shares do not match what $\mathcal{S}$ computed for that committee, then $\mathcal{S}$ sets **should-abort** $\leftarrow 1$. Similarly, if any outer shares corresponding to $|\mathcal{H}_k| \leq d'$ do not match what $\mathcal{S}$ computed, then $\mathcal{S}$ sets **should-abort** $\leftarrow 1$.
- 8. Finally, if **should-abort** = 1, then $\mathcal{S}$ sends **abort** to $\mathcal{F}_{\text{RandVerif}}$.
- 9. Otherwise, for inner sharings in which $|\mathcal{T}_k| \leq d'$ and thus $|\mathcal{H}_k| > d'$, $\mathcal{S}$ sends the corresponding reconstructed corrupted parties' shares to $\mathcal{F}_{\text{RandVerif}}$. For any inner sharings in which $|\mathcal{T}_k| > d'$, $\mathcal{S}$ sends the corresponding honest parties' shares, along with the reconstructed outer share (based on the outer shares of those in which $|\mathcal{H}_k| > d'$), to $\mathcal{F}_{\text{RandVerif}}$.

Now we show that the distributions of the ideal world and real world are identical. It is clear that all communication during the protocol is distributed identically in both worlds, since $\mathcal{S}$ follows the protocol exactly for honest parties. Thus, we just need to check that the output of the honest parties is the same in both worlds. We have two cases:

- 1. $\mathcal{S}$ sends **abort** to $\mathcal{F}_{\text{RandVerif}}$ in the ideal world. In this case, we need to show that the honest parties output **abort** in the real world. One way in which $\mathcal{S}$ aborts is if some combined sharing $\llbracket \mathbf{r}_{j^*}^* \rrbracket_{d,d'}$ is not of the right degree; which will be caught by $\mathcal{F}_{\text{DV}}$ in the real world.

$\mathcal{S}$ can also abort if the adversary sends different shares to $\mathcal{F}_{\text{DV}}$ on behalf of the corrupted parties than what $\mathcal{S}$ computed; in the real world, this is also caught with probability 1 by honest input parties due to the error-detection properties of the secret sharing scheme.

- 2. $\mathcal{S}$ does not send **abort** to $\mathcal{F}_{\text{RandVerif}}$ in the real world. In this case, we need to argue that the honest parties' shares of $\llbracket \mathbf{r} \rrbracket_{d,d'}$ are identical in the real and ideal worlds. First, for committees in which $|\mathcal{T}_k| > d'$, the honest parties' inner shares are distributed identically, since $\mathcal{S}$ sends directly to $\mathcal{F}_{\text{RandVerif}}$ these shares.

For the rest, first note that there is at least one honest input party P_i . Their shared virtual sharings $\llbracket \mathbf{r}_{i,j} \rrbracket_{d,d'}$ for $j \in [\ell]$ will be random, and thus so will each $\llbracket \mathbf{r}_j \rrbracket_{d,d'}$ for $j \in [\ell]$.

Now, consider any $d + 1 - |\mathcal{T}_{\text{virt}}|$ honest committees. In the real world, the inner sharings of these committees are distributed randomly, conditioned on the corrupted parties' shares and the sharing being of degree d' . $\mathcal{F}_{\text{RandVerif}}$

samples these sharings in the same way and thus they are distributed the same way in the ideal world. For every other honest committee the corresponding outer share is determined by the outer shares of the above $d + 1 - |\mathcal{T}_{\text{virt}}|$ honest committees along with those of the corrupted committees; this is the same in both worlds. Moreover, their inner sharings are distributed randomly, conditioned on the corrupted parties' shares, the determined outer share, and the sharing being of degree d' ; this too is identical in both worlds. $\square$

B.4 Inputs

In this section, we formally describe the ideal functionality which takes the input of correct and corrupt parties and prepares virtual secret shares from them.

Functionality 8: $\mathcal{F}_{\text{Input}}$

1. For each input party $P_i \in I$:
 - (a) If P_i is corrupted, receive whole sharing $\llbracket x_i \rrbracket_{d,d'}$ from the adversary; if it is not of the proper degree, **abort**.
 - (b) Otherwise, receive from the honest input party x_i and from the adversary the corrupted party's shares of $\llbracket x_i \rrbracket_{d,d'}$. Sample the rest of the sharing consistent with the above.
2. If there is no abort from corrupted parties, output to the honest parties their shares of all inputs.

Protocol 12: Π_{Input}

1. Each input party P_i secret shares its input x_i to create $\llbracket x_i \rrbracket_{d,d'}$. They distribute the shares to the other parties.
2. The parties invoke $\mathcal{F}_{\text{DV}}$ on all of the input sharings $\llbracket x_i \rrbracket_{d,d'}$, aborting if it does.

Lemma 12. Π_{Input} UC-realizes $\mathcal{F}_{\text{Input}}$ in the $\mathcal{F}_{\text{DV}}$ -hybrid model.

Proof. The simulation is straightforward: In the case of corrupted input party, $\mathcal{S}$ receives the honest parties' shares and can reconstruct $\llbracket x_i \rrbracket_{d,d'}$. If any are of the wrong degree, $\mathcal{S}$ sends $\mathcal{F}_{\text{Input}}$ abort when emulating $\mathcal{F}_{\text{DV}}$, otherwise it sends the sharing to $\mathcal{F}_{\text{Input}}$. For honest parties, $\mathcal{S}$ can just send both to the adversary and $\mathcal{F}_{\text{Input}}$ random corrupted parties' shares. By the privacy of the virtual secret sharings, this is a good simulation. $\square$

C Standard Procedures for Π_{Main}

C.1 Multiplication

The first procedure we need is an efficient multiplication procedure. For this, we use the standard “king” technique of [DN07], presented in π_{Mult} .

Procedure 13: π_{Mult}

Parameters: This protocol is parametrized by default position vector β and position vector pos .

1. Let $\llbracket \mathbf{x} \rrbracket_{d,d'}, \llbracket \mathbf{y} \rrbracket_{d,d'}$ be the input virtual secret sharings.
2. The parties invoke π_{Rand} twice to prepare random virtual sharings $\llbracket \mathbf{r} \rrbracket_{d,d'}$ and $\llbracket \mathbf{r} \rrbracket_{m-1,\eta-1}$ with the same underlying secret vector $\mathbf{r}$.
3. The parties locally compute $\llbracket \mathbf{d} \rrbracket_{m-1,\eta-1} \leftarrow \llbracket \mathbf{x} \rrbracket_{d,d'} \cdot \llbracket \mathbf{y} \rrbracket_{d,d'} + \llbracket \mathbf{r} \rrbracket_{m-1,\eta-1}$.
4. The parties send their shares of $\llbracket \mathbf{d} \rrbracket_{m-1,\eta-1}$ to party P_1 .
5. Party P_1 reconstructs $\mathbf{d}$ then computes $\llbracket \mathbf{d} \rrbracket_{\rho-1,0}$, and distributes the shares to all parties.
6. All parties locally compute $\llbracket \mathbf{z} \rrbracket_{d,d'} \leftarrow \llbracket \mathbf{d} \rrbracket_{\rho-1,0} - \llbracket \mathbf{r} \rrbracket_{d,d'}$.^a
7. The parties invoke $\pi_{\text{SharingTrans}}$ on $\llbracket \mathbf{z} \rrbracket_{d,d'}$ with (β, pos, I) , where I is the identity map to obtain $\llbracket \mathbf{z} \rrbracket_{\text{pos}}|_{\text{pos}}$ and output it.^b

^a These are the sharings that will later be checked in $\pi_{\text{CheckMult}}$

^b $\llbracket \mathbf{z} \rrbracket_{d,d'}$ can be interpreted as $\llbracket \mathbf{z} \rrbracket_{m-1,\eta-1}$, since its underlying polynomials are of degree at most $m-1$ and $\eta-1$, respectively.

The communication complexity of the above protocol is $O(\sqrt{n}|I|)$ amortized elements at the beginning of Π_{Main} for $\pi_{\text{SharingTrans}}$ and π_{Rand} (note, all such invocations of π_{Rand} for multiplications can be done at the beginning of the main MPC protocol, Π_{Main}), and $O(n)$ total for the rest.

Checking Multiplications. At the end of the MPC protocol, we will have to check that all multiplication gates were performed correctly. For this, we use the technique of [CGH⁺18], in which the parties sample a random sharing $\llbracket \mathbf{r} \rrbracket_{d,d'}$ at the beginning of the protocol (before the input phase), and in addition to computing a sharing of every wire in the circuit, $\llbracket \mathbf{w} \rrbracket_{d,d'}$, they also compute a version that is randomized by $\mathbf{r}$, $\llbracket \mathbf{r} \cdot \mathbf{w} \rrbracket_{d,d'}$. At the end of the protocol they check that indeed $\llbracket \mathbf{r} \cdot \mathbf{w} \rrbracket_{d,d'}$ shares the secret of $\llbracket \mathbf{w} \rrbracket_{d,d'}$, multiplied by $\mathbf{r}$. The intuition here is that given some multiplication of $\llbracket \mathbf{x} \rrbracket_{d,d'}$ and $\llbracket \mathbf{y} \rrbracket_{d,d'}$ resulting in $\llbracket \mathbf{z} \rrbracket_{d,d'}$, if the adversary causes the output sharing to share $\mathbf{z} = \mathbf{x} \cdot \mathbf{y} + \delta$, for some $\delta \neq \mathbf{0}$, then the adversary needs to ensure that $\llbracket \mathbf{r} \cdot \mathbf{z} \rrbracket_{d,d'}$ indeed shares $\mathbf{r} \cdot (\mathbf{x} \cdot \mathbf{y} + \delta)$. However, since $\mathbf{r}$ is random and unknown to the adversary, the adversary would have to guess $\mathbf{r}$ to do so, which can only happen with negligible probability.

Procedure $\pi_{\text{CheckMult}}$ below checks all of the multiplication gates simultaneously, using a random linear combination.

Procedure 14: $\pi_{\text{CheckMult}}$

1. Let $\{(\llbracket \mathbf{z}_i \rrbracket_{d,d'}, \llbracket \mathbf{r} \cdot \mathbf{z}_i \rrbracket_{d,d'})\}_{i \in [\ell]}$ be the set of multiplications to be checked, $\llbracket \mathbf{r} \rrbracket_{d,d'}, \llbracket \mathbf{s} \rrbracket_{d,d'}$ input random sharings ($\mathbf{r}$ is the same as in the previous sharings), and χ the input random element.

2. The parties compute $\llbracket \mathbf{u} \rrbracket_{d,d'} \leftarrow \sum_{i=0}^{\ell} \chi^i \cdot \llbracket \mathbf{r} \cdot \mathbf{z}_i \rrbracket_{d,d'}$ and $\llbracket \mathbf{v} \rrbracket_{d,d'} \leftarrow \sum_{i=0}^{\ell} \chi^i \cdot \llbracket \mathbf{z}_i \rrbracket_{d,d'}$.
3. The parties invoke $\mathcal{F}_{\text{Open}}$ on $\llbracket \mathbf{r} \rrbracket_{d,d'}$ to get r .
4. The parties compute $\llbracket \mathbf{w} \rrbracket_{d,d'} \leftarrow \llbracket \mathbf{u} \rrbracket_{d,d'} - r \cdot \llbracket \mathbf{v} \rrbracket_{d,d'}$.
5. The parties invoke π_{Mult} on $\llbracket \mathbf{s} \rrbracket_{d,d'}$ and $\llbracket \mathbf{w} \rrbracket_{d,d'}$ to obtain $\llbracket \mathbf{t} \rrbracket_{d,d'}$.
6. The parties invoke $\mathcal{F}_{\text{Open}}$ on $\llbracket \mathbf{t} \rrbracket_{d,d'}$ to get $\mathbf{t}$.
7. If $\mathbf{t} \neq \mathbf{0}$, the parties abort.

The communication complexity of $\pi_{\text{CheckMult}}$ is $\tilde{O}(n^{1.5})$ for $\mathcal{F}_{\text{Open}}$ (the cost of π_{Mult} is amortized across the circuit size $|C|$).

Lemma 13. *If for any $i \in [\ell]$, the secret r of $\llbracket \mathbf{r} \rrbracket_{d,d'}$, $\mathbf{z}_i$ of $\llbracket \mathbf{z}_i \rrbracket_{d,d'}$ and ζ_i of $\llbracket \mathbf{r} \cdot \mathbf{z}_i \rrbracket_{d,d'}$ are not such that $r \cdot \mathbf{z}_i = \zeta_i$, then the honest parties will abort with all-but-negligible probability at the end of $\pi_{\text{CheckMult}}$.*

Proof. Let index $j \in [\ell]$ be such that $\zeta_j = r \cdot \mathbf{z}_j + \delta_j$, for some $\delta_j \neq \mathbf{0}$, and in particular, $\delta_{j,k} \neq 0$. Let $F(\mathbf{y})$ be the degree $\ell - 1$ polynomial $F(\mathbf{y}) = \sum_{i=1}^{\ell} (r \cdot \mathbf{z}_{j,k} - r \cdot \zeta_{j,k}) \cdot \mathbf{y}^i$. Since $\delta_{j,k} \neq 0$, this is a nonzero polynomial. Thus, by the Schwartz-Zippel Lemma, since the evaluation point χ of this polynomial is random, $F(\chi) \neq 0$ with all-but-negligible probability and thus $\llbracket \mathbf{w} \rrbracket_{d,d'}$ is nonzero. In this case, since $\llbracket \mathbf{s} \rrbracket_{d,d'}$ is random, $\llbracket \mathbf{t} \rrbracket_{d,d'}$ will be random, and thus the parties will abort. Indeed, even though the adversary can introduce some additive error so that $\mathbf{t} \neq \mathbf{w} * \mathbf{s}$, it can only guess this error to force $\mathbf{t} = \mathbf{0}$ with negligible probability. $\square$

C.2 Network Routing

As described in the Sharing Transformation section (Section 7), for each layer of the circuit, the parties need to be able to perform *network routing*, i.e., prepare virtual sharings $\llbracket \mathbf{x} \rrbracket_{d,d'}$ and $\llbracket \mathbf{y} \rrbracket_{d,d'}$ for every batch of gates in that layer. We use the following procedure, π_{Network} inspired by [GPS22] to do so. It first uses multiplication by public unit vectors $\mathbf{e}_i \in \mathbb{F}^p$, which are all zeros except for a single one in the i -th position, to individually collect each single wire that will be used in $\llbracket \mathbf{x} \rrbracket_{d,d'}$ from the previous layers. It then sums these individual collected sharings up to obtain a sharing with all of the needed wires, and finally performs a sharing transformation to get these secret wire values in their correct corresponding positions and shared with the proper degree. Note that some gates in the circuit will have constant inputs; π_{Network} also collects these constant inputs into the output sharings.

Procedure 15: π_{Network}

Parameters: This procedure is parameterized by the default packed secret positions β .

1. Let $x_1, x_2, \dots, x_{\ell_1}$ be the different wire values from previous layers and $c_1, c_2, \dots, c_{\ell_2}$ be the constant values that the parties want to collect in a new sharing $\llbracket \mathbf{x} \rrbracket_{d,d'}$. Let $\mathbf{x}' = (x_1, \dots, x_{\ell_1}, c_1, \dots, c_{\ell_2}, 0, \dots, 0) \in \mathbb{F}^\rho$.
2. For $i \in [\ell_1]$, let $\llbracket \mathbf{x}^{(i)} || \mathbf{pos}^{(i)} \rrbracket_{d,d'}$ be the virtual packed sharing from some previous layer that contains the secret x_i at position p_i . Let $p_{\ell_1+1}, \dots, p_\rho$ be the first $\rho - \ell_1$ distinct positions that are different from $p_1, \dots, p_{\ell_1}$ and $\alpha_1, \dots, \alpha_m$. Let $\mathbf{pos} = (p_1, \dots, p_\rho)$.
3. Let $\mathbf{e}_i$ be the i -th unit vector in $\mathbb{F}^\rho$. All parties locally (and deterministically) compute a virtual packed sharing $\llbracket \mathbf{e}_i || \mathbf{pos} \rrbracket_{\rho-1,0}$.
4. All parties locally compute $\llbracket \mathbf{x}' || \mathbf{pos} \rrbracket_{d+\rho-1,d'} \leftarrow \sum_{i=1}^{\ell_1} \llbracket \mathbf{e}_i || \mathbf{pos} \rrbracket_{\rho-1,0} \cdot \llbracket \mathbf{x}^{(i)} || \mathbf{pos}^{(i)} \rrbracket_{d,d'} + \sum_{i=1}^{\ell_2} c_i \cdot \llbracket \mathbf{e}_{\ell_1+i} || \mathbf{pos} \rrbracket_{\rho-1,0}$.
5. Let $f : \mathbb{F}^\rho \rightarrow \mathbb{F}^\rho$ be the linear map such that $\mathbf{x} = f(\mathbf{x}')$. The parties invoke $\pi_{\text{SharingTrans}}$ on $\llbracket \mathbf{x}' || \mathbf{pos} \rrbracket_{d+\rho-1,d'}$ with $(\mathbf{pos}, \beta, f)$ to obtain $\llbracket \mathbf{x} \rrbracket_{d,d'}$.^a
6. The parties output $\llbracket \mathbf{x} \rrbracket_{d,d'}$.

^a $\llbracket \mathbf{x}' || \mathbf{pos} \rrbracket_{d+\rho-1,d'}$ can be interpreted as $\llbracket \mathbf{x}' || \mathbf{pos} \rrbracket_{m-1,\eta-1}$, since its underlying polynomials are of degree at most $m-1$ and $\eta-1$, respectively.

The Communication complexity of π_{Network} is $O(\sqrt{n}|I|)$ amortized at the beginning of Π_{Main} for $\pi_{\text{SharingTrans}}$ and $O(n)$ total for the rest.

D Supplementary Material for MPC Construction

D.1 Proof of Lemma 3

Proof. We start with a simulator $\mathcal{S}$, which initializes $\text{should-abort} \leftarrow 0$:

1. $\mathcal{S}$ first emulates $\mathcal{F}_{\text{RandVerif}}$ by sampling random sharings $\llbracket \mathbf{r}_j \rrbracket_{d,d'}$ consistent with the corrupted shares received from the adversary. It is easy to see that the secrets of these sharings alone determine the resulting random vector $\mathbf{s}$ (via the appropriate linear combinations).
2. $\mathcal{S}$ then computes the sharings $\llbracket \mathbf{s}^j \rrbracket_{d+\rho-1,d'}$ and $\llbracket \mathbf{f}(\mathbf{s})^j \rrbracket_{d+\rho-1,d'}$ (only the honest parties' shares of corrupted inner sharings) for $j \in [\ell]$.
3. $\mathcal{S}$ then simulates π_{CommOpen} :
 - (a) To simulate π_{CommRand} , for each honest input party P_j , $\mathcal{S}$ samples random virtual sharing $\llbracket \mathbf{r}_j \rrbracket_{d+\rho-1,d'}$ and for each $k \in [\rho]$, random sharings for committee $\mathcal{C}_i$, $[\mathbf{r}_{j,k}]_\delta$, then sends to the adversary the corrupted parties' shares.
 - (b) $\mathcal{S}$ receives the honest parties' shares of the same from the adversary and can reconstruct the corresponding secret vector $\mathbf{r}_j$ of $\llbracket \mathbf{r}_j \rrbracket_{d+\rho-1,d'}$.
 - (c) If $\mathcal{C}_i \in \mathcal{H}_{\text{virt}}$, $\mathcal{S}$ can reconstruct the whole sharings $[\mathbf{r}_{j,k}]_\delta$ for $k \in [\rho]$, in the case that $\delta = d'$. If it is not of the correct degree, $\mathcal{S}$ sets $\text{should-abort} \leftarrow 1$.
 - (d) $\mathcal{S}$ can then compute the sharing $\llbracket \mathbf{r} \rrbracket_{d+\rho-1,d'}$ (only honest parties' inner shares of corrupted committees).

- (e) Next, if party P_1 is corrupted:
- i. $\mathcal{S}$ sends to the adversary the honest parties' shares of $\llbracket e \rrbracket_{d+\rho-1,d'}$
 - ii. $\mathcal{S}$ then receives back from the adversary e on behalf of each honest party P in committee $\mathcal{C}_i$.
- Otherwise:
- i. $\mathcal{S}$ receives from the corrupted parties their shares of $\llbracket e \rrbracket_{d+\rho-1,d'}$, then reconstructs it and sends it to the adversary if there are corrupted parties in committee $\mathcal{C}_i$.
- (f) In either case, for $\mathcal{C}_i \in \mathcal{H}_{\text{virt}}$, for each $k \in [\rho]$, $\mathcal{S}$ can compute the committee's outer shares $\mathbf{f}_k(s_k)^i$, and thus the whole sharings $\llbracket \mathbf{f}'(s_k) || \mathbf{pos}'_k \rrbracket_{d,d'}$, including the secrets (only honest parties' shares of corrupted committees). If the degree of any outer sharing is greater than d , $\mathcal{S}$ sets **should-abort** $\leftarrow 1$. $\mathcal{S}$ can also compute for each $k \in [\rho]$, $\varepsilon_k \leftarrow f'(s_k) - f(s_k)$. Finally, $\mathcal{S}$ can also compute the honest parties' inner shares of $\llbracket \mathbf{s} \rrbracket_{m-1,\eta-1}$ and therefore also the vector Δ corresponding to corrupted parties' shares that combine with the honest parties' shares to yield $\mathbf{s}$.
4. Then, $\mathcal{S}$ emulates $\mathcal{F}_{\text{DV}}$ by receiving the corrupted parties' shares from the adversary; if any are not the same as what $\mathcal{S}$ computed, $\mathcal{S}$ sets **should-abort** $\leftarrow 1$.
5. If **should-abort** = 1, $\mathcal{S}$ sends $\mathcal{F}_{\text{DV}}$ abort; otherwise, $\mathcal{S}$ sends to $\mathcal{F}_{\text{DV}}$ the corrupted parties' shares of $(\llbracket s_k || \mathbf{pos}_k \rrbracket_{d,d'}, \llbracket \mathbf{f}'(s_k) || \mathbf{pos}'_k \rrbracket_{d,d'})$, ε_k for each $k \in [\rho]$, and Δ .

Now we show that the ideal and real worlds are identically distributed. It is clear that the emulation of $\mathcal{F}_{\text{RandVerif}}$ and simulation of π_{CommOpen} is identical to the real world. Thus it remains to show that the output distributions (including the output of $\mathcal{F}_{\text{DV}}$) are identical. We start with the case in which there is an abort in the ideal world. The first way in which this can happen is if the degree of the output sharings is too high; $\mathcal{F}_{\text{DV}}$ will catch this in the real world. This can also happen if the adversary provides the wrong shares to $\mathcal{F}_{\text{DV}}$, which would also be caught by $\mathcal{F}_{\text{DV}}$ in the real world, due to the error detection properties of the sharing scheme.

If there is not an abort, first, the sharings output by $\mathcal{F}_{\text{RandVerif}}$ are random and unknown to the adversary. Also, in π_{CommOpen} , since each honest input party's $\llbracket \mathbf{r}_j \rrbracket_{d+\rho-1,d'}$ is random, so too is $\llbracket \mathbf{r} \rrbracket_{d+\rho-1,d'}$ and thus so too is $\llbracket e \rrbracket_{d+\rho-1,d'}$. Moreover, a similar argument can be made about the honest committees' sharings $\llbracket \mathbf{r}_j \rrbracket_\delta$. Therefore, nothing is revealed to the adversary about any of the honest inner sharings of the output sharings, and therefore any of the honest outer shares, all of which are distributed randomly constrained to the corrupted shares and the mapping between the secrets. Furthermore, it is clear that $\mathcal{S}$ can completely determine $\mathbf{s}$ from the sharings $\llbracket \mathbf{r}_j \rrbracket_{d,d'}$ and reconstruct $\llbracket \mathbf{f}'(s_k) || \mathbf{pos}'_k \rrbracket_{d,d'}$, thus its computation of the errors ε_j and corrupted parties' contributions Δ are correct. Putting the above together, the real world output distribution matches the ideal world output distribution, since $\mathcal{F}_{\text{MapRand}}$ samples the output sharings randomly based on the δ_j 's, corrupted parties' shares, and Δ . $\square$

D.2 Proof of Theorem 5

Proof. We begin with a simulator $\mathcal{S}$, which starts by setting $\text{error} \leftarrow 0$:

1. $\mathcal{S}$ first emulates $\mathcal{F}_{\text{CSamp}}$ by sampling random committees $\mathcal{C}_1, \dots, \mathcal{C}_m$ and sending them to the adversary.
2. $\mathcal{S}$ then emulates $\mathcal{F}_{\text{RandVerif}}$ to sample random sharings $\llbracket \mathbf{r} \rrbracket_{d,d'}, \llbracket \mathbf{s} \rrbracket_{d,d'}$ consistent with corrupted parties' shares received from the adversary.
3. For each corrupted input party P_i , $\mathcal{S}$ emulates $\mathcal{F}_{\text{Input}}$ by receiving from the adversary the entire input sharing $\llbracket \mathbf{x}_i \rrbracket_{d,d'}$. For each honest input party P_i , $\mathcal{S}$ emulates $\mathcal{F}_{\text{Input}}$ by receiving from the adversary the corrupted parties' shares of the input sharing $\llbracket \mathbf{x}_i \rrbracket_{d,d'}$.
4. To simulate any invocation of $\pi_{\text{SharingTrans}}$:
 - (a) $\mathcal{S}$ will emulate $\mathcal{F}_{\text{MapRand}}$ by sampling random sharings $\llbracket \mathbf{r} || \mathbf{pos} \rrbracket_{m-1, \eta-1}$ and $\llbracket \mathbf{f}(\mathbf{r}) + \varepsilon || \mathbf{pos} \rrbracket_{d,d'}$ consistent with corrupted parties' shares received from the adversary.
 - (b) Then, if P_1 is corrupted, $\mathcal{S}$ sends to the adversary the honest parties' shares of $\llbracket \mathbf{r} || \mathbf{pos} \rrbracket_{m-1, \eta-1}$ and receives back $\llbracket \mathbf{f}(\mathbf{e}) || \mathbf{pos}' \rrbracket_{\rho-1, 0}$; since the corrupted outer shares are known by $\mathcal{S}$, it can compute the difference δ between what it should have received and what it did receive. If P_1 is honest, $\mathcal{S}$ receives the corrupted parties' shares from the adversary, reconstructs $\mathbf{e}$ and sends back $\llbracket \mathbf{f}(\mathbf{r}) || \mathbf{pos}' \rrbracket_{\rho-1, 0}$; it can again compute δ with the same information.
5. For any invocation of π_{Mult} :
 - (a) To simulate π_{Rand} , $\mathcal{S}$ samples $\llbracket \mathbf{r}_i \rrbracket_{d,d'}, \llbracket \mathbf{r}_i \rrbracket_{m-1, \eta-1}$ on behalf of the honest input parties P_i and receives back shares of the same from the adversary for corrupted input parties.
 - (b) Then if P_1 is corrupted, $\mathcal{S}$ sends to the adversary the honest parties' shares of $\llbracket \mathbf{r} \rrbracket_{m-1, \eta-1}$ and receives back $\llbracket \mathbf{d} \rrbracket_{\rho-1, 0}$; since $\mathcal{S}$ knows the corrupted parties' shares, it can compute the difference γ between what it should have received and what it did receive. If P_1 is honest, $\mathcal{S}$ receives the corrupted parties' shares from the adversary, and sends back $\llbracket \mathbf{r} \rrbracket_{\rho-1, 0}$, for random $\mathbf{r}$; again, it can compute γ with the same information.
 - (c) It simulates $\pi_{\text{SharingTrans}}$ as above.
6. For any invocation of π_{Network} , $\mathcal{S}$ simply simulates $\pi_{\text{SharingTrans}}$ as above.
7. Note that for every multiplication gate, the inputs are collected via sharing transformation, then multiplied using π_{Mult} . We thus have additive errors $\delta_x - \varepsilon_x$ for the $\mathbf{x}$ input, additive errors $\delta_y - \varepsilon_y$ for the $\mathbf{y}$ input, and additive errors γ_z for the output $\mathbf{z}$; if it is not the case that $(\mathbf{x} + \delta_x - \varepsilon_x) * (\mathbf{y} + \delta_y - \varepsilon_y) + \gamma_z = \mathbf{x} * \mathbf{y}$, then $\mathcal{S}$ sets $\text{error} \leftarrow 1$. (Note, if the inputs came from addition gates, then actually more sharing transformations are involved)
8. For the output phase:
 - (a) To emulate $\mathcal{F}_{\text{RG}}$, $\mathcal{S}$ samples a random element χ and sends it to the adversary.
 - (b) To emulate $\mathcal{F}_{\text{DV}}$, first, if P_1 was corrupted and shared some $\llbracket \mathbf{f}(\mathbf{e}) || \mathbf{pos}' \rrbracket_{\rho-1, 0}$ of the wrong degree, then $\mathcal{S}$ sends **abort** to $\mathcal{F}_{\text{MPC}}$. If not, then $\mathcal{S}$ receives

the corrupted parties' shares of each $\llbracket f(e) \parallel \mathbf{pos}' \rrbracket_{\rho-1,0}$ from the adversary; if any disagree with what $\mathcal{S}$ computed, $\mathcal{S}$ sends **abort** to $\mathcal{F}_{\text{MPC}}$ (note that $\mathcal{S}$ can easily keep track of all corrupted parties' shares).

- (c) To simulate $\pi_{\text{CheckMult}}$:
 - i. $\mathcal{S}$ first emulates $\mathcal{F}_{\text{Open}}$ by sending random r to the adversary.
 - ii. $\mathcal{S}$ simulates π_{Mult} as above.
 - iii. Finally, $\mathcal{S}$ emulates $\mathcal{F}_{\text{Open}}$ as follows. If $\text{error} = 0$, then $\mathcal{S}$ sends $\mathbf{0}$ to the adversary; otherwise it sends $\mathbf{s}$ to the adversary and then **abort** to $\mathcal{F}_{\text{MPC}}$.
- (d) Finally, $\mathcal{S}$ receives each output z from $\mathcal{F}_{\text{MPC}}$ and emulates $\mathcal{F}_{\text{Open}}$ by sending z to the adversary.

We now show the ideal and real worlds are indistinguishable. The simulations of mask-and-open steps in $\pi_{\text{SharingTrans}}$ and π_{Mult} respectively are perfect, since the random sharings $\llbracket \mathbf{r} \parallel \mathbf{pos} \rrbracket_{m-1,\eta-1}$ and $\llbracket \mathbf{r} \rrbracket_{m-1,\eta-1}$, respectively, are random and unknown to the adversary. Similarly, for $\pi_{\text{CheckMult}}$, since $\llbracket \mathbf{s} \rrbracket_{d,d'}$ is random and unknown to the adversary, so too will $\llbracket \mathbf{t} \rrbracket_{d,d'}$ (even with additive errors from the adversary).

Thus it remains to show that the output distributions are indistinguishable. Clearly, if there is no abort in the ideal world, the outputs are identical. There are two ways in which $\mathcal{S}$ aborts in the ideal world: either from the emulation of $\mathcal{F}_{\text{DV}}$ or from $\text{error} = 1$. The former case is perfectly handled by $\mathcal{F}_{\text{DV}}$. The latter case is handled according to Lemma 13.

To calculate the communication complexity of Π_{Main} , we use $\tilde{O}(|I|(n+\kappa)+n^{1.5})$ bits for $\mathcal{F}_{\text{CSamp}}$, $\tilde{O}(n^{1.5})$ elements for $\mathcal{F}_{\text{DV}}$, $\mathcal{F}_{\text{RG}}$, and $\pi_{\text{CheckMult}}$, $O(n \cdot |I|)$ elements for $\mathcal{F}_{\text{RandVerif}}$, $O(\sqrt{n} \cdot |I|)$ amortized across the entire circuit C and $O(n)$ for each layer of the circuit for π_{Mult} , $\pi_{\text{SharingTrans}}$, and π_{Network} , and $\tilde{O}(n)$ elements amortized per output gate from $\mathcal{F}_{\text{Open}}$. $\square$