Towards Scalable YOSO MPC via Packed Secret-Sharing

Daniel Escudero[1], Elisaweta Masserova[2], and Antigoni Polychroniadou[1]

[1]J.P. Morgan AI Research & J.P. Morgan AlgoCRYPT CoE
[2]Carnegie Mellon University

Abstract

The YOSO (You Only Speak Once) model, introduced by Gentry et al. (CRYPTO 2021), helps to achieve strong security guarantees in cryptographic protocols for distributed settings, like blockchains, with large number of parties. YOSO protocols typically employ smaller anonymous committees to execute individual rounds of the protocol instead of having all parties execute the entire protocol. After completing their tasks, parties encrypt protocol messages for the next anonymous committee and erase their internal state before publishing ciphertexts, thereby enhancing security in dynamically changing environments.

In this work, we consider the problem of secure multi-party computation (MPC), a fundamental problem in cryptography and distributed computing. We assume honest majority among the committee members, and work in the online-offline, i.e., preprocessing, setting. In this context, we present the first YOSO MPC protocol where efficiency—measured as communication complexity—*improves* as the number of parties *increases*. Specifically, for $0 < \epsilon < 1/2$ and an adversary corrupting $t < n(\frac{1}{2} - \epsilon)$ out of n parties, our MPC protocol exhibits enhanced scalability as n increases, where the online phase communication becomes *independent* of n. Prior YOSO MPC protocols considered t as large as $(n-1)/2$, but a significant hurdle persisted in obtaining YOSO MPC with communication that does not scale linearly with the number of committee members, a challenge that is exagerbated when the committee size was large per YOSO's requirements. We show that, by considering a small "gap" of $\epsilon > 0$, the sizes of the committees are only marginally increased, while online communication is significantly reduced.

Furthermore, we explicitly consider fail-stop adversaries, i.e., honest participants who may inadvertently fail due to reasons such as denial of service or software/hardware errors. In prior YOSO work, these adversaries were grouped with fully malicious parties. Adding explicit support for them allows us to achieve even better scalability.

1 Introduction

Secure multiparty computation (MPC) enables a set of mutually distrusting parties to securely compute a function of their inputs while only interacting among each other, revealing only the output of the function. Security holds even if t out of the n parties are corrupted and collude. Unfortunately, MPC often requires communication-intensive interactive protocols, and this is particularly critical in large-scale distributed environments such as blockchains. This situation quickly led to adopting the notion of *committees*, already common in distributed computing.[1] Instead of a large pool of parties performing MPC, a subset of the parties—a *committee*—is the one in charge of executing the MPC protocol on behalf of the larger set. The intuition is that, by setting parameters properly, if the

[1]A good example is seminal Bracha's committees idea, which is widely used in many protocols today. [9]

large set of N parties has T corruptions, then with high probability the smaller set of n parties will have t corruptions, where the ratio t/n is only *slightly larger* than the original T/N.

Unfortunately, committee-based MPC suffers from an inherent drawback: it is insecure when an adversary can corrupt parties *after* the committee has been sampled, which is a property known as *adaptive security*. For example, if $N = 1000$, we assume the adversary can corrupt $T = 400$ parties, and we sample committees of size $n = 400$, an adaptive adversary can choose to corrupt *all* of the parties of a selected committee once it learns their identities.

Player replaceability, which was introduced by Chen and Micali [16] in the context of obtaining consensus in the Algorand blockchain, is a beautiful way to address this shortcoming. Player-replaceable protocols also utilize committees, which can be tasked with, e.g., proposing and agreeing on blockchain block. However, what differentiates player-replaceable protocols from prior work, is that each committee member sends only a *single message*, and is completely *anonymous* until they have done so. The YOSO (*You Only Speak Once*) model, introduced by Gentry et al. [29], provided a clean formalization of such player-replaceable protocols.

In more detail, YOSO distinguishes between physical machines (which can potentially retain state long-term) and stateless *roles*, which are deployed on demand to perform a certain task. YOSO further separates the protocol design from the *role-assignment* functionality that selects which physical machine is performing which role. Assuming that role-assignment is secure, i.e., the adversary is unable to predict which machine will be executing which role, YOSO schemes can withstand even highly powerful adversaries, e.g. an adversary who is performing an adaptive Denial of Service attack. Simultaneously, the committees which are performing the computation are much smaller than the overall pool of physical machines. This enables MPC protocols with communication complexity that scales sublinearly with the total number of parties, and protocols remain secure even if the adversary can compromise large fractions of physical machines (up to a third or even half, depending on the role-assignment).

The efficiency of YOSO MPC. Since its inception, many works have expanded the limits of the YOSO model, *cf.* [7, 40, 15, 36]. Very interesting results exist, and we survey some of them in detail in Section 1.2. However, we note a common trend shared by all YOSO MPC protocols to date: their total communication complexity grows as the size of the committees increases. Currently, the most asymptotically efficient YOSO MPC scheme in the computational setting is that by [29], and it achieves $O(n^2)$ communication per gate (for wide circuits it can be further amortized to $O(n)$). This is particularly harmful for YOSO, where the committee sizes are considered very large. To illustrate this, consider the work of [29], which makes use of the role assignment from [6] to sample committees. In [6] the authors show that if the global corruption ratio is $f = 0.25$, then committees of size roughly 40k are needed to get honest majority! The associated YOSO MPC protocol given in [29] has a communication complexity that scales linearly with the size of the committee, leading to extremely poor performance for committees as large as 40K.

1.1 Our Contributions

We explore YOSO MPC when the committees not only have an honest majority, but there is a *gap* proportional to the committee size between the number of honest and the number of corrupt parties. In more detail, let n be the committee size, and t be the amount of corruptions per committee. Instead of studying the case $n = 2t + 1$, which is the traditional setting in YOSO MPC, we initiate the study of YOSO MPC for $t < n(\frac{1}{2} - \epsilon)$, for some constant $\epsilon > 0$. Our motivation is two-fold:

1. As we show, when $t < n(\frac{1}{2} - \epsilon)$ it is possible to design YOSO MPC protocols that scale much better as n grows, in contrast to the case when $\epsilon = 0$. In particular, it is possible to obtain an

online phase whose total communication is *independent* of the committee size n, allowing for better scalability.

2. For large number of parties—the context that YOSO is aimed at—it is not unreasonable to assume that the adversary cannot corrupt not only 49.99% (1/2) of the parties, but a smaller percentage such as, say 45% or 40% $(1/2 - \epsilon)$. In fact, for committee-based protocols such as YOSO, we show that requiring the committees to have a gap $\epsilon > 0$ only increases committee sizes by a marginal amount, while reducing online communication drastically.

3. Finally, even though traditional YOSO MPC uses $n = 2t + 1$, due to the tail bound, for the corruption threshold of the global set must anyway hold $T < N(\frac{1}{2} - \epsilon')$. Thus, in the grand scheme of things, we are simply increasing an ϵ' that is already there.

We elaborate on these points below.

1.1.1 Improved communication of YOSO MPC.

We present a YOSO MPC protocol that is divided in two phases. In an offline phase, a series of *preprocessing committees* generate certain correlations that can be consumed by subsequent committees. Here, our communication is asymptotically the same as prior works: $O(n|C|)$. Our main benefits appear in the online phase, which is set in motion once the inputs to the computation are known. Here, our communication is $O(|C|)$, *independent of the committee size* n. In more detail, we make use of packed secret-sharing in order to improve prior protocols by a factor of $k \approx n \cdot \epsilon$, at the expense of a slightly larger committee size. Note that the bigger the ϵ, which corresponds to less corruption tolerance, the more the efficiency gains.

1.1.2 Role assignment for committees with "gap".

We observe that prior role-assignment works such as [6] focus on choosing the committee size n so that the new corruption threshold t satisfies $t < n/2$, and YOSO MPC protocols such as [29] are designed assuming committees of this size, with 1/2 as the corruption ratio. In order to obtain committees whose corruption ratio is $1/2 - \epsilon$, we generalize the probability analysis from [6] and show that, by choosing committees of *slightly* larger size, we can achieve a smaller corruption ratios of $1/2 - \epsilon$, for some $\epsilon > 0$. Our results are discussed in Section 6. Crucially, we show that the cost of enabling the gap $\epsilon > 0$ is really minimal, and its benefits are substantial. For example, for 5% global corruptions we can already get 28× improvement by moving from committees of size 900 to 1000. For larger corruption ratios such as 20%, we can get 1000× online improvement with respect to the approach from [6, 29] by moving from committees of size ≈ 18k to ≈ 20k.

Remark 1 (Fail-stop tolerance). We also highlight an important benefit of considering the ratio $1/2 - \epsilon$ instead of 1/2. All current YOSO solutions are designed to tolerate certain amount of *active* corruptions, i.e., corrupted parties can behave arbitrarily maliciously. However, in large-scale settings, it's essential to safeguard against not only active attacks but also fail-stop parties—honest participants who may inadvertently fail due to various reasons, including external attacks like denial of service, software/hardware errors, or natural events. In current YOSO MPC protocols fail-stop parties are treated the same as active corruptions, which has been shown to be an overkill in the non-YOSO literature [26, 35, 3, 21]. Considering a ratio of $1/2 - \epsilon$ not only allows us to gain efficiency, but it also allows us to tolerate unresponsive honest parties. We show in Section 5.4 that, if we cut by a factor of two the gains in communication, we can tolerate $n\epsilon$ honest parties who may become unresponsive during the protocol execution. This can happen perhaps due to crashes or other issues, which is essential for large scale settings such as YOSO MPC.

1.2 Related work

The first YOSO MPC schemes have been proposed by Gentry et al. [29]. Their information-theoretic scheme is based on the famous BGW protocol [5], which, as the authors note, is essentially already a YOSO protocol in the semi-honest setting, where corrupt parties follow the protocol, but may try to learn extra information by observing the protocol execution. Unfortunately, the communication complexity of the extension of this protocol to the fully-malicious setting in the YOSO model is prohibitively high. The computationally secure solution of Gentry et al. is based on the CDN protocol [18] and uses a a linearly homomorphic threshold encryption scheme, where the public key is known to everyone, and the secret key is shared among the committee members. The circuit is evaluated gate-by-gate, with the addition gates being computed locally. To perform multiplications, parties can decrypt partial results using their shares of the global threshold key. One remaining open problem was the generation of the setup, i.e., the generation of the threshold public key and sharing of the secret key shares to the first committee. This was addressed by Braun et al. [10], who showed how to obtain distributed key generation for a linearly homomorphic threshold encryption scheme in the YOSO setting. Using this primitive allows to easily obtain a CDN-style solution for YOSO MPC. Concurrently to the work of Braun et al., Kolby et al. [36] introduced constant-round YOSO MPC protocols without setup based on garbled circuits and threshold fully homomorphic encryption. Kolby et al. further proposed (a non-constant) CDN-based protocol, which they use to obtain the (constant-round) setup of one of their other constructions.

YOSO Role-Assignment. The task of assigning physical machines to roles, along with a mechanism of sending private messages to these roles is a core building block in YOSO constructions. The seminal work of Benhamouda et al. [6] introduced the first such mechanism – *receiver-anonymous communication channels* (RACCs). Benhamouda et al.'s solution allows protocol participants to generate short-term keys for the next committee members, with the public portion of the key known to everyone, and the secret portion known only to the corresponding machine. This is done *without* revealing which machines are selected to participate in the next committee. Later, Gentry et al.[30] introduced another RACCs solution. Their protocol supported higher adversarial thresholds, but at the cost of being much more computationally expensive than the solution of Benhamouda et al. More recently, Campanelli et al. [11] proposed an *Encryption to the Future* primitive, a paradigm of sending messages to the anonymous committees in the future. Their solution is based on a special kind of witness encryption. Cascudo et al. [15] proposed a solution based on publicly verifiable secret sharing (PVSS) towards anonymous committees. Finally, Canetti et al. [13] recently formalized a functionality which captures a broad class of role-assignment protocols.

Almost stateless MPC The works on Scales MPC [1, 2] consider an interesting relaxation of YOSO, where the clients are allowed to speak more than once.

In addition to YOSO, the Fluid model [17] has been recently introduced by Choudhuri et al. Among the Fluid works, the protocol of Bienstock et al. [8] achieves linear communication complexity, but provides only security with abort. Deligios et al. and David et al. proposed Fluid protocols with guaranteed output delivery (GOD), albeit at the cost of high communication complexity [22, 24, 23]. Rachuri and Scholl explored Fluid in the dishonest majority setting [41]. Finally, David et al. [22] introduced a clean abstraction of MPC on layered graphs, which captures a stringent setting at the intersection of Fluid and YOSO MPC.

The work of Goyal et al. [32] designs MPC with GOD in which parties speak only once under the assumption of conditional storage and retrieval systems (works of Benhamouda et al. [6] and Goyal et al. [31] can serve as instantiations of such systems).

Finally, recent work in the model of YOSO with worst-case corruptions [39, 38, 37] focuses on a specialized MPC functionality: distributed randomness generation. In this YOSO variant, committees are executed in sequence, and the adversary can corrupt a total of t committees.

2 Our Model and YOSO Background

We consider a synchronous model of execution, i.e., the protocol proceeds in rounds, and parties are aware in which round they are currently in. We consider a rushing adversary, i.e., the adversary can see the messages sent by the honest roles before producing messages of the corrupt roles. We consider committees of n parties, and the adversary can corrupt at most t parties in each committee.

Additionally, we recap the background on YOSO. It is a variation of the UC framework [12] which separates between physical machines and roles that these machines play in the protocol. Roles do not keep private state between rounds and each role sends only a single message. It is the job of the role assignment functionality to map roles to the machines. The YOSO MPC protocols are then described entirely using roles, we refer to such protocols as "abstract YOSO" (vs. "natural YOSO" which includes explicit role assignment) in the following. We now recap further details of the YOSO model and its differences to the standard UC. We defer the formalization of YOSO broadcast to Appendix C. For a detailed version, refer to the original YOSO work [30].

- To ensure that roles speak only once, the framework "yoso-ifies" them with a YOSO wrapper, which ensures that roles are killed immediately after they have spoken. This is modelled by a Spoke token which ideal functionalities send to roles (the time at which the functionality sends the token is determined by the functionality itself). Upon obtaining Spoke, the role also passes it onto its sub-routines and its environment. Once a role is killed, the machine executing it also erases any associated state, which prevents the adversary from obtaining any information from the roles that have already spoken by corrupting the corresponding machines.
- The roles have access to idealised communication functionalities, which in particular allows point-to-point messages between roles.
- Gentry et al. [30] note that if the adversary does not know which roles are assigned to a machine before it is corrupted, the "best" that an adversary can do is corrupt machines at random. They further make the observation that for such random corruptions, the difference between adaptive corruptions and static corruptions is minimal. This allows to design protocols secure against a somewhat restricted adversary in the abstract YOSO model, which nevertheless translate into adaptively secure protocols in the natural YOSO model.
- While the roles which perform the computation are typically corrupt at random, the input and output nodes are assumed to be known machines and are subject to chosen corruptions.

We denote a yoso-ified role R by $\mathsf{YoS}(\mathsf{R})$, and we denote the protocol obtained by yoso-ifying all roles in protocol π by $\mathsf{YoS}(\pi)$. We say that π YOSO-securely implements $\mathcal{F}$ and write $\pi \leq^{YOSO} \mathcal{F}$, if $\mathsf{YoS}(\pi)$ UC-securely implements $\mathcal{F}$ against the given class of controlled environments. Recall that π UC-securely implements $\mathcal{F}$, if for all PPT adversaries $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that $\mathsf{Real}_{\pi,\mathcal{A},\mathcal{E}} \approx \mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ for all PPT environments $\mathcal{E}$. Here, $\mathsf{Real}_{\pi,\mathcal{A},\mathcal{E}}$ denotes the random variable representing $\mathcal{E}$'s output in the real world, where the adversary $\mathcal{A}$ is interacting with the honest parties who execute π; and $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ denotes the random variable representing $\mathcal{E}$'s output in the ideal world, where the simulator $\mathcal{A}$ is interacting with the ideal functionality $\mathcal{F}$.

Finally, we define YOSO MPC. We recall the ideal functionality $\mathcal{F}_{\mathsf{MPC}}^{F}$ as defined by Gentry et al. The functionality distinguishes between two stages that the protocol goes through, GettingInputs and Evaluated. While honest parties give input in the first round, the protocol can be in the GettingInputs stage for a long time, as corrupted parties might only be committed to their inputs at a later point in time. Once the adversary S decides that it is time to give outputs, it sets the stage to Evaluated. We distinguish between input roles $\mathsf{Role}^{\mathsf{In}}$, output roles $\mathsf{Role}^{\mathsf{Out}}$, and roles $\mathsf{Role}^{\mathsf{Cmp}}$ which perform the computation. Malicious denotes the set of fully malicious roles, Leaky the set of honest but curious roles, and Honest the set of honest roles.

Ideal Functionality $\mathcal{F}_{\mathsf{MPC}}^{F}$ for MPC

The functionality is wrapping a function $F(x_1, \ldots, x_n) \to (y_1, \ldots, y_m)$. Roles in $\mathsf{Role}^{\mathsf{In}}$ hold inputs and roles in $\mathsf{Role}^{\mathsf{Out}}$ receive outputs.

- Initially set the stage to be GettingInputs. We set a default input for all roles, which they may overwrite later: for all $\mathsf{R} \in \mathsf{Role}^{\mathsf{In}}$ let $x_{\mathsf{R}} = 0$.
- On input $(\mathsf{Input}, \mathsf{R} \in \mathsf{Role}^{\mathsf{In}}, x \in \mathsf{Msg})$ proceed as follows:
  1. Store $x_{\mathsf{R}} = x$.
  2. If $\mathsf{R} \in \mathsf{Honest}$ then output $(\mathsf{Input}, \mathsf{R}, |x|)$ to $\mathcal{S}$.
  3. If $\mathsf{R} \in \mathsf{Leaky} \cup \mathsf{Malicious}$ output $(\mathsf{Input}, \mathsf{R}, x)$ to $\mathcal{S}$.

  If R is honest then output Spoke to R. If R is honest then consider only the first input, and only if it is given in round 1.
- On Evaluated from $\mathcal{S}$ in a round $r > 1$ and when the stage is GettingInputs, set the stage to be Evaluated and compute $\{y_{\mathsf{R}}\}_{\mathsf{R} \in \mathsf{Role}^{\mathsf{Out}}} = F(\{x_{\mathsf{R}}\})_{\mathsf{R} \in \mathsf{Role}^{\mathsf{In}}}$. Store y_{R} for all $\mathsf{R} \in \mathsf{Correct}$ and output to $\mathcal{S}$ the value $\{y_{\mathsf{R}}\}_{\mathsf{R} \in \mathsf{Role}^{\mathsf{Out}} \cap (\mathsf{Malicious} \cup \mathsf{Leaky})}$.
- On input $(\mathsf{Read}, \mathsf{R} \in \mathsf{Role}^{\mathsf{Out}})$, if the stage is Evaluated output y_{R} to R.

The YOSO MPC is then defined as follows:

Definition 1 (YOSO MPC). We say that π YOSO-securely realizes F against τ corruption if $\pi \leq^{YOSO} \mathcal{F}_{\mathsf{MPC}}^{F}$ for the set of environments allowed to corrupt any number of roles in $\mathsf{Role}^{\mathsf{In}} \cup \mathsf{Role}^{\mathsf{Out}}$ and a uniformly random fraction τ of $\mathsf{Role}^{\mathsf{Cmp}}$.

3 Technical Overview – Improving Computational YOSO MPC

We now give an overview of our YOSO MPC, for formal details see Section 5. The state-of-the-art protocol boasts an amortized communication complexity of $O(n)$ per circuit gate [29] assuming the circuit's width is $O(n)$. Using the widely used offline/online paradigm, we attain $O(1)$ online communication complexity and $O(n)$ offline communication complexity per gate, maintaining the assumption that the circuit width is also $O(n)$. We use bold font to represent a vector.

3.1 Starting idea: Building upon an Efficient Non-YOSO Protocol in the Offline/Online Paradigm

Our starting point is the Turbopack [25] protocol, which achieves MPC with abort with constant online communication complexity in the traditional (non-YOSO) setting. We now review the main techniques used in this work. At a high level, Turbopack follows the common MPC approach of secret-sharing the clients' secrets, and then evaluating the circuit gate-by-gate. To achieve its constant amortized communication, Turbopack utilizes *packed* secret sharing, where $k \in O(n)$ secrets are stored using the same sharing. Intuitively, this ensures that the cost of evaluating a set of k multiplication gates is the same as evaluating only a single multiplication gate.

Same as in the other works which utilize packing, the main challenge in Turbopack is to solve what is known as the *network routing issue*. Specifically, at each circuit layer, the secrets within a packed secret sharing may not be in correct order. For instance, the packed vectors may not be correctly aligned, or sharings intended for a set of multiplication gates are scattered among many different packed output sharings of the previous layer. Naively, re-organizing the secrets at each level by first reconstructing, and then placing them in correct positions of the packed vectors would result in communication complexity of $O(n)$, thus defying the purpose of using packing.

Instead, Turbopack uses a *circuit-dependent* preprocessing phase to prepare correlated randomness. This later helps to solve the routing issue efficiently. During this phase, the circuit is known, but the parties' inputs are not. Turbopack assigns a value λ^α to each circuit wire α, such that:

- For any output wire of an input gate and any output wire of a multiplication gate, λ^α is uniformly random.
- For any addition gate γ with input wires α and β, $\lambda^\gamma = \lambda^\alpha + \lambda^\beta$.

After the preprocessing, for each batch $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$ of k input and multiplication gates, each party has a packed share of $\lambda^{\boldsymbol{\alpha}} = (\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$.

In the online phase, Turbopack uses these sharings to step-by-step compute $\mu^\alpha = v^\alpha - \lambda^\alpha$ in clear for each gate α, where v^α is the actual value on this wire. The addition gates can be computed locally, as $\mu^\gamma = \mu^\alpha + \mu^\beta = v^\alpha + v^\beta - \lambda^\alpha - \lambda^\beta$, where μ^α and μ^β are known, and each party has a share of λ^α and λ^β from the preprocessing. For the multiplication gates, Turbopack adapts the technique of packed Beaver triples [33], a generalization of the famous Beaver triple technique [4]. Here, the computation again crucially relies on the parties knowing the preprocessed shares of $\lambda^{\boldsymbol{\alpha}}$.

To obtain a solution which is compatible with the YOSO model, we need to solve the following:

- In YOSO, parties who participate in the online phase of the protocol are *not* the same as the ones preparing the preprocessed values, since parties change in every round. We must ensure that the parties of the online committee obtain the preprocessed values, e.g., shares of $\lambda^{\boldsymbol{\alpha}}$, in a way that does not break security, as the adversary can corrupt parties both in the online phase and also during the preprocessing. More crucially, in YOSO we do not know during the offline phase the identities of the parties who will execute the online phase, so we must find a way to pass the secret values to these roles into the future *without assuming knowledge of their YOSO role keys*, all while ensuring that the communication complexity remains efficient.
- Finally, Turbopack achieves security with abort, while our goal is to obtain a solution which achieves GOD. One might think that moving to a computational setting and utilizing non-interactive zero-knowledge proofs ought to be sufficient for GOD. However, in order to obtain its low communication complexity, Turbopack uses a trick which seems inherently incompatible with GOD. Intuitively, shares of μ are revealed only to a *single* party. Thus, a single corruption

might prevent the protocol from finishing. To ensure that our solution achieves GOD, we must find another way to compute μ_γ, while retaining efficient communication complexity.

3.2 YOSO-ifying Turbopack via CDN

To solve these issues, we carefully combine the techniques from Turbopack with the ideas of the famous CDN protocol [18], and utilize additional tricks to ensure that the amortized online communication complexity remains constant per gate.

CDN relies on a system-wide public key tpk of a linearly homomorphic threshold encryption scheme, where the corresponding secret key tsk is shared among the committee members. In our case, the committees will be changing, hence we will refresh the shares of the tsk after each usage.

Ours is not the first work to propose using CDN in the YOSO setting – both the original YOSO work [29], as well as the work by Braun et al. [10] use a CDN-based approach. Indeed, CDN is very appealing in the YOSO setting: with private communication between the committees being one of the bottlenecks in YOSO MPC, CDN-based approaches require only a *small* secret state to be maintained, i.e., the parties' shares of the secret key tsk. However, the communication complexity in both works remains high due to the following. To compute a multiplication gate with encrypted inputs x and y, both works simply generate Beaver triples (a, b, c) encrypted under the tpk on the fly, and let committee members use their shares of tsk to decrypt the ciphertexts $x + a$ and $y + b$. During the decryption, committee members must not only compute and publish their share of the decrypted value, but also pass the (refreshed) shares of tsk to the next committee. Naively, this results in $O(n^2)$ communication per gate. If committees are responsible for $O(n)$ gates, the communication complexity can be brought down to amortized $O(n)$ by letting committees process $O(n)$ gates in parallel. However, further amortization is *not possible*, as to decrypt each ciphertext that is encrypted under tpk, we ultimately need $O(n)$ committee members to supply their shares.

As we will see, in our protocol, by building upon Turbopack, during the online phase **(1)** we will not have to compute Beaver triples, and, more importantly, **(2)** we will perform the expensive threshold decryption procedure only once. Instead, we use the threshold encryption primarily for computations during the offline phase, as well as for passing preprocessed secret values, e.g., shares of $\boldsymbol{\lambda^\alpha}$, to online parties in an efficient way.

Keys For Future: YOSO-compatible Preprocessing Usage. Specifically, to pass preprocessed secret values to the members of the online committees (without knowing their YOSO role keys), we propose the following approach of making traditional preprocessing YOSO-friendly. First, during the setup we generate *keys for future* (KFF) – these are the keys that act as substitutes for the YOSO role keys of the future roles. We generate a KFF public and secret key pair for each member of the later committee. We publish the public keys, and encrypt secret keys under the threshold public key tpk of the linearly homomorphic threshold encryption scheme. Whenever a party needs to encrypt certain information to a future committee role whose role key is not known yet, the party encrypts it under the KFF public key of that specific role. Later, once the protocol reaches the phase where the role keys of the corresponding committees are known, one committee can use their shares of tsk to decrypt the ciphertexts containing the KFF secret keys, and re-encrypt these secret keys under the YOSO role key of the corresponding role. Note that using KFF, as opposed to simply encrypting every secret value that needs to be passed to a future role under the tpk is crucial – each value that is reconstructed via the CDN's decryption procedure requires at least $O(n)$ amortized communication complexity, as $O(n)$ committee members must publish their shares in order to reconstruct this value. Assuming that a role processes $O(n)$ gates, using KFF results in *constant* online complexity per gate, as opposed to the linear complexity of the naive approach.

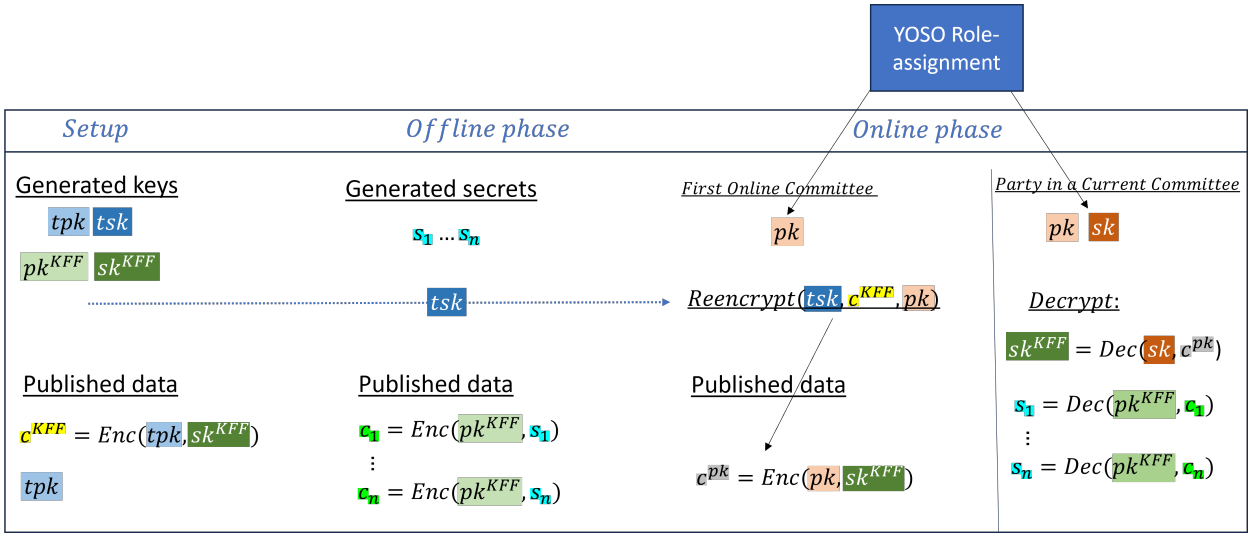


Figure 1: Key usage in different phases of our YOSO MPC.

We now describe the stages of our protocol – setup, offline, and online phase, in reverse order, and give intuition for the challenges we encounter and our approaches to solving these.

Notation and Packed Shamir Secret Sharing Before diving in, we briefly specify the notation we use in the following, as well as recall the *packed* Shamir secret sharing scheme [27], which is a generalization of the standard Shamir secret sharing [42]. Intuitively, it allows to secret-share a batch of secrets using a single Shamir sharing. For a vector $x \in \mathbb{F}^k$, we use $[\![\boldsymbol{x}]\!]_d$ to denote a degree-d packed Shamir sharing, where $k-1 \leq d \leq n-1$. It requires $d+1$ shares to reconstruct the whole sharing, and any $d-k+1$ shares are independent of the secrets.

Same as the standard Shamir secret sharing, the packed version is linearly homomorphic, i.e., for all $d \geq k-1$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\![\boldsymbol{x}+\boldsymbol{y}]\!]_d = [\![\boldsymbol{x}]\!]_d + [\![\boldsymbol{y}]\!]_d$. Further, we can perform multiplication on the shares as follows: For all $d_1, d_2 \geq k-1$ subject to $d_1 + d_2 < n$, and for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\![\boldsymbol{x} * \boldsymbol{y}]\!]_{d_1+d_2} = [\![\boldsymbol{x}]\!]_{d_1} * [\![\boldsymbol{y}]\!]_{d_2}$. Finally, we note that packed Shamir sharing is *multiplication-friendly* [33], in the sense that when $d \leq n-k$, all parties can locally multiply a public vector $\boldsymbol{c} \in \mathbb{F}^k$ with a degree-d packed Shamir sharing $[\![\boldsymbol{x}]\!]_d$ as follows:

1. All parties first locally compute a degree-$(k-1)$ packed Shamir sharing of $\boldsymbol{c}$, denoted by $[\![\boldsymbol{c}]\!]_{k-1}$. Note that for a degree-$(k-1)$ packed Shamir sharing, all shares are determined by the secrets.

2. All parties then locally compute $[\![\boldsymbol{c} * \boldsymbol{x}]\!]_{n-1} = [\![\boldsymbol{c}]\!]_{k-1} * [\![\boldsymbol{x}]\!]_{n-k}$.

In the following, we denote the above process by $[\![\boldsymbol{c} * \boldsymbol{x}]\!]_{n-1} = \boldsymbol{c} * [\![\boldsymbol{x}]\!]_{n-k}$.

3.3 Online Phase

During the online phase, we follow Turbopack's approach of efficiently computing $\mu^\alpha = v^\alpha - \lambda^\alpha$ in clear for each circuit wire α. Here, v^α is the actual value on the wire, and λ^α is the value assigned to this wire during the preprocessing. We now describe how to obtain μ^α for each gate type.

In order to compute μ^α for the inpute wire α of a circuit, where this input is supplied by some client C, we let C learn λ^α. Then, C can publish $\mu^\alpha = v^\alpha - \lambda^\alpha$. To compute μ^γ for an output wire γ of an addition gate with input wires α and β, anyone can simply add μ^α and μ^β.

Computing multiplication gates is trickier. To obtain a communication-efficient solution, Turbopack adapts the technique of packed Beaver triples [33], which are triples $([\![\boldsymbol{a}]\!]_d, [\![\boldsymbol{b}]\!]_d, [\![\boldsymbol{c}]\!]_d)$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are random vectors from $\mathbb{F}^k$ and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. Intuitively, for a group of multiplication gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, it holds that

$$\begin{aligned}\boldsymbol{\mu^\gamma} = \boldsymbol{v^\alpha} * \boldsymbol{v^\beta} - \boldsymbol{\lambda^\gamma} &= (\boldsymbol{\mu^\alpha} + \boldsymbol{\lambda^\alpha}) * (\boldsymbol{\mu^\beta} + \boldsymbol{\lambda^\beta}) - \boldsymbol{\lambda^\gamma} \\ &= \boldsymbol{\mu^\alpha} * \boldsymbol{\mu^\beta} + \boldsymbol{\mu^\alpha} * \boldsymbol{\lambda^\beta} + \boldsymbol{\lambda^\alpha} * \boldsymbol{\mu^\beta} + \boldsymbol{\lambda^\alpha} * \boldsymbol{\lambda^\beta} - \boldsymbol{\lambda^\gamma} \quad (1)\end{aligned}$$

Thus, to compute $\boldsymbol{\mu^\gamma}$, we must let parties of the current online committee obtain shares of $\boldsymbol{\lambda^\alpha}, \boldsymbol{\lambda^\beta}$ and shares of $\boldsymbol{\Gamma^\gamma} = \boldsymbol{\lambda^\alpha} * \boldsymbol{\lambda^\beta} - \boldsymbol{\lambda^\gamma}$. Looking ahead, these values will be generated during the preprocessing. To efficiently pass them to the members of the online committee *without* assuming that the YOSO role keys of the online committees are known during the preprocessing phase, we use our keys for future (see 3.2 for details). Briefly, the secret shares of $\boldsymbol{\lambda^\alpha}, \boldsymbol{\lambda^\beta}$, and $\boldsymbol{\Gamma^\gamma}$ are encrypted under the keys for future, and the secret key parts of the keys for future are in turn encrypted under the tpk. To give parties their secret shares, during the online phase we let the first committee use their shares of tsk to re-encrypt the secret shares of $\boldsymbol{\lambda^\alpha}, \boldsymbol{\lambda^\alpha}$, and $\boldsymbol{\Gamma^\gamma}$ under the (now known) YOSO role keys of the online committee members. After this point we will not require access to tsk anymore. Hence, there is no need to re-share shares of tsk, which allows us to save communication.

Efficiently Computing $\boldsymbol{\mu_\gamma}$ with GOD. One issue remains. Note that in Turbopack $\boldsymbol{\mu_\gamma}$ is computed by having each party compute its share of the equation above locally, and send the result to a single party P_1. This ensures that the communication complexity remains low – each out of n parties only sends one message to P_1, and since $\boldsymbol{\mu_\gamma}$ covers to $O(n)$ gates, this ensures that amortized communication complexity per gate is still constant. This unfortunately does not work for us, as our goal is to obtain a solution which has the guaranteed output delivery property. To alleviate this, we first observe that, in all currently known YOSO works, P2P messages are sent by posting encryptions to future committees in, for example, a bulletin board. However, what is more interesting is that *broadcast messages* are also disseminated in the same way. This means that, in YOSO MPC, **broadcast has effectively the same cost as P2P communication**! Interestingly, a (slightly weaker) statement is inherent to the model: As physical participants of the next committees are not known beforehand, messages must be communicated to everyone. Hence, one-to-one communication costs effectively the same as one-to-all. While in our online protocol such one-to-all communication is sufficient, in the following for simplicity we will not distinguish between one-to-all and broadcast (as all currently known YOSO schemes anyway use broadcast for one-to-one communication). We are able to leverage this observation to improve the communication of YOSO MPC by using less reserved use of the broadcast channel. Additionally, in order to ensure that μ_γ is reconstructed correctly, roles who are contributing their local shares will prove that they did the computation correctly. For this, as in previous works, we will have parties compute and publish a NIZK proof of correctness.

3.4 Offline Phase

During the offline phase, for each batch of k multiplication gates our goal is to prepare shares of correctly routed packed wire values $\boldsymbol{\lambda^\alpha}$, as well as $\boldsymbol{\Gamma^\gamma}$. To achieve this, for each multiplication gate, we first prepare a Beaver triple. Then, we let the current committee members jointly generate

a random value λ^α for each output wire of an input/multiplication gate α, encrypted under the global threshold public key tpk (in the following, denote such ciphertext for the wire α by c^α). Note that for all values that are published during the offline phase, we let parties attach NIZK proofs explaining that they did everything correctly. We then use only those values for the computation, for which the corresponding proofs verify.

Given ciphertexts c^α, we process the circuit gate by gate (first without using packing). For the addition gates, we simply compute the sum of two ciphertexts containing the random wire values of the two input wires to this gate. For each multiplication gate with encrypted inputs c^α and c^β, and encrypted output wire value c^γ, we need to compute the encryption of $\Gamma_\gamma = \lambda^\alpha * \lambda^\beta - \lambda^\gamma$. For this, we follow the approach of Gentry et al. [29]. Specifically, to compute a multiplication gate with encrypted inputs c^α and c^β, we consume one of the Beaver triples (a, b, c), and let committee members use their shares of tsk to decrypt the ciphertexts $c^\alpha + a$ and $c^\beta + b$.

Next, in order to ensure that our online phase is efficient, we pack the random wire values $(\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$ for each group of k circuit input wires $(\alpha_1, \ldots, \alpha_k)$ which are supplied by a single client. Similarly, we pack the random vector $\boldsymbol{\lambda^\alpha} = (\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$ of the output wires for each batch of k multiplication gates. This is done as follows: Given ciphertexts $c^{\lambda^{\alpha_1}}, \ldots, c^{\lambda^{\alpha_k}}$, we first generate t *additional* encryptions of random values $c^{r_1}, \ldots, c^{r_t}$. Then, we use these ciphertexts to homomorphically compute encryptions of the evaluation points of the polynomial $f(x)$ of degree $t + k - 1$, which on each x-point $-(i-1)$ evaluates to λ_i^α, $i \in [k]$, and on i evaluates to r_i, $i \in [t]$. Then, we use all points (both the values λ^{α_i}, $i \in [k]$ and extra random values r_i, $i \in [1, \ldots, t]$), to locally interpolate using the homomorphic properties of the threshold encryption scheme, and this way obtain *packed* shares $f(1), \ldots, f(n)$ of $\boldsymbol{\lambda^\alpha}$ which are encrypted under tpk.

Finally, currently the packed shares are encrypted under the tpk. This is problematic: If a share is encrypted under tpk, then during the online phase one committee will spend $O(n)$ communication to decrypt a *single packed share*, with each committee member publishing its CDN-style share of the packed share. This would defy the usage of packing and result in $O(n^2)$ communication per packed sharing, hence linear amortized cost—no better than prior works. To solve this, we re-encrypt the packed shares to the KFFs of the roles in the online committee who will use these shares. It allows us to pay the linear cost during the offline phase, and keep the online phase very efficient.

3.5 Role Assignment with "Gap"

Finally, we recall that prior works like [29] focused on assigning roles in such a way that an adversary corrupts at most $49.9\ldots\%$ of the parties in each committee. They provide a method known as *cryptographic sortition*, and determine the appropriate parameters to ensure this corruption bound in the committees. In Section 6 we show how to use their approach to sample committees that ensure a lower adversarial bound (say 45%, for example), which allows us to use our YOSO MPC protocol. We carefully adapt their analysis and quantitatively show that lowering the amount of corrupted parties corresponds to choosing slightly larger committees. We then show that, even if committee size increase marginally, the benefits of the reduced communication of our protocol quickly kick in.

4 Preliminaries

We now describe our building blocks. We take large parts the following verbatim from Gentry et al. [29].

4.1 Linearly Homomorphic Key Rerandomizable Threshold Encryption

A linearly homomorphic (over ring $\mathbb{R}$) key rerandomizable threshold encryption scheme TE has the following algorithms:

- $\mathsf{TKGen}(1^\kappa) \to (tpk, tsk_1, \ldots, tsk_n)$: An algorithm that, given the security parameter κ, sets up the public key tpk and the shares $tsk_1, \ldots, tsk_n$ of the secret key.
- $\mathsf{TEnc}(tpk, m; r) \to \beta$: An algorithm that, given the public key, a message $m \in \mathbb{R}$ and randomness r, outputs an encryption β of m.
- $\mathsf{TPDec}(tpk, tsk_i, \beta) \to d_i$: An algorithm that, given the public key, a share tsk_i of the secret key and a ciphertext β, outputs a partial decryption d_i.
- $\mathsf{TDec}(tpk, \{d_i\}_{i \in S, |S| > t}) \to m$: An algorithm that, given sufficiently many partial decryptions, returns the decrypted message m.
- $\mathsf{TEval}(tpk, \beta_1, \ldots, \beta_k, \lambda_1, \ldots, \lambda_k) \to \beta$: A deterministic algorithm that, given the public key, ciphertexts $\beta_1, \cdots, \beta_k$ corresponding to messages $m_1, \ldots, m_k \in \mathbb{R}^k$ and coefficients $\lambda_1, \ldots, \lambda_k \in \mathbb{R}^k$, outputs a ciphertext β that encrypts $\sum_{i=1}^k \lambda_i m_i \in \mathbb{R}$.
- $\mathsf{TKRes}(tpk, tsk_i; r_i) \to (m_{i,1}, \ldots, m_{i,n})$: An algorithm that, given the public key and a share of a secret key, produces n messages to help with the rerandomization of the secret key sharing.
- $\mathsf{TKRec}(tpk, \{m_{j,i}\}_{j \in S, |S| > t}) \to tsk_i$: An algorithm that, given sufficiently many messages for the rerandomization of the secret key sharing, outputs a share of the secret key.
- $\mathsf{SimTPDec}(tpk, \beta, m, \{tsk_i\}_{i \in [n] \setminus S}, \{d_i\}_{i \in S}) \to \{d_i\}_{i \in [n] \setminus S}$: A simulation algorithm that, given a ciphertext, a target message, and partial decryptions belonging to corrupt parties, simulates partial decryptions belonging to honest parties that cause TDec to output the desired message.

Such encryption scheme must satisfy the following correctness properties:

- Decryption on honestly produced ciphertext and keys returns the appropriate message.
- Decryption remains correct after homomorphic evaluation.
- Decryption remains correct after a rerandomization of the secret key sharing.

These properties are intuitive, and we do not formalize them here. The important security property of a TE scheme is *partial decryption simulatability*, described in Appendix A.1.

We can instantiate such an encryption scheme by Shamir sharing a Paillier decryption key [19], see Gentry et al. [29] for details.

4.2 Non-Interactive Zero-Knowledge Arguments of Knowledge

A non-interactive zero-knowledge argument of knowledge ($\mathsf{NIZKAoK}$) scheme has the following algorithms, as described by Groth and Maller [34].

$\mathsf{Setup}(1^\kappa, \mathcal{R}) \to (crs, td)$: An algorithm that, given the security parameter, sets up the global common reference string crs and the trapdoor td for the NIZKAoK system.

$\mathsf{P}(crs, \phi, w) \to \pi$: An algorithm that, given the common reference string crs for a relation $\mathcal{R}$, a statement ϕ and a witness w, returns a proof π that $(\phi, w) \in \mathcal{R}$. $\mathsf{V}(crs, \phi, \pi) \to$ **accept/reject**:

An algorithm that, given the common reference string crs for a relation $\mathcal{R}$, a statement ϕ and a proof π, checks whether π proves the existence of a witness w such that $(\phi, w) \in \mathcal{R}$.

$\mathsf{SimP}(crs, td, \phi) \to \pi$: An algorithm that, given the common reference string crs for a relation R, the trapdoor td and a statement ϕ, simulates a proof of the existence of a witness w such that $(\phi, w) \in R$.

Security properties A NIZKAoK scheme must be correct (that is, verification using an honestly produced proof must return accept). We formally describe the NIZKAoK security properties (zero knowledge, knowledge soundness, and simulation extractability) in Appendix A.2.

5 Improving Computational YOSO MPC

In this section we describe our YOSO MPC protocol in the computational setting. It consists of three phases – setup, offline, and online phase. Importantly, we only assume the knowledge of the YOSO role-assignment public keys for the committee of a given phase starting *only* with the first committee of that phase. In particular, during the offline phase we do not assume knowledge of the public keys of the roles of the future online phase.

In the following, let κ denote the security parameter. Let C_i denote the i's online committee, C_i^{Off} denote the i's offline committee, and $C_{i,j}$ denote the j's role of the online committee C_i (similarly for $C_{i,j}^{\mathsf{Off}}$). Let n denote the size of the committee. For a set S, let $|S|$ denote the size of this set. Let $\mathsf{TE} = (\mathsf{TKGen}, \mathsf{TKEnc}, \mathsf{TPDec}, \mathsf{TDec}, \mathsf{TEval}, \mathsf{TKRes}, \mathsf{TKRec})$ denote a linearly homomorphic key rerandomizable threshold encryption scheme, let $\mathsf{NIZKAoK} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V}, \mathsf{SimP})$ denote a simulation extractable non-interactive zero-knowledge argument of knowledge, and let $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ denote an additively homomorphic public key encryption scheme.

For the ease of presentation, we describe each phase of the protocol separately. In the following, we use bold font to represent a vector.

5.1 Setup

During the setup, we generate the keys for future (KFF), which allow us to efficiently pass messages to future roles whose role keys are not known during the earlier stages of the execution. We further generate the setup for the NIZK proof system that we will later use to prove correctness of the protocol execution. Finally, we set up the linearly homomorphic key rerandomizable threshold encryption. Note that we assume that in our encryption scheme the public key corresponds to a single decryption key. We describe our assumed setup in Figure 5.1 below.

$\Pi_{\mathsf{YOSO\text{-}Setup}}$

1. Generate *keys for future*[a]:
   - For each role $C_{l,i}$ of each committee C_l participating in the online phase:
     - Generate $(pk_{C_{l,i}}^{\mathsf{KFF}}, sk_{C_{l,i}}^{\mathsf{KFF}}) \leftarrow \mathsf{PKE.Gen}(1^\kappa)$.
     - Let $c_{C_{l,i}}^{\mathsf{KFF}} \leftarrow \mathsf{TEnc}(tpk, sk_{C_{l,i}}^{\mathsf{KFF}})$.
   - For each input-contributing party P_i:
     - Generate $(pk_{P_i}^{\mathsf{KFF}}, sk_{P_i}^{\mathsf{KFF}}) \leftarrow \mathsf{PKE.Gen}(1^\kappa)$.
     - Let $c_{P_i}^{\mathsf{KFF}} \leftarrow \mathsf{TEnc}(tpk, sk_{P_i}^{\mathsf{KFF}})$.

2. Generate the NIZK setup via $crs \leftarrow \mathsf{NIZKAoK.Setup}(1^\kappa)$.

3. Generate the threshold public key and secret keys as $(tpk, tsk_1, \ldots, tsk_n) \leftarrow \mathsf{TKGen}(1^\kappa)$. Publish tpk and give tsk_i to $C_{1,i}^{\mathsf{Off}}$.

[a] These are **not** the keys that will be later generated by the YOSO role-assignment for these roles.

5.2 Offline phase

During the offline phase, our goal is to prepare correlated randomness that will be later consumed during the online phase. Specifically, we generate the following:

- For each input wire α that is supplied by client P_i, we generate a random value λ^α.
- For each output wire α of a multiplication gate, we generate a random value λ^α. Then, for each group of k multiplication gates, we prepare a packed sharing of the vector $\boldsymbol{\lambda^\alpha} = (\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$.
- For each group of k multiplication gates with input wires $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)$, and output wires $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_k)$, we compute a packed sharing of the vector $\boldsymbol{\Gamma^\gamma} = \boldsymbol{\lambda^\alpha} * \boldsymbol{\lambda^\beta} - \boldsymbol{\lambda^\gamma} = (\lambda^{\alpha_1} \cdot \lambda^{\beta_1} - \lambda^{\gamma_1}, \ldots, \lambda^{\alpha_k} \cdot \lambda^{\beta_k} - \lambda^{\gamma_k})$.

We then prepare the following data for the online phase:

- For each input wire α that is supplied by client P_i, we prepare a ciphertext c^α, which encrypts λ^α under the KFF of P_i.
- For each group of k multiplication gates with output wires $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_k)$, we prepare an encryption of the i-th packed share of $\boldsymbol{\lambda^\alpha}$ under the public key of the i-th committee member of the online phase who will need this value.
- Similarly, for each group of k multiplication gates with output wires $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_k)$, we prepare an encryption of the i-th packed share of $\boldsymbol{\Gamma^\gamma}$ under the public key of the i-th committee member of the online phase who will need this value.

The offline phase works as follows: First, for each multiplication gate, we prepare a Beaver triple. Then, we let each committee member generate a random value λ_i^α for each output wire of an input/multiplication gate, and encrypt it under the global threshold public key tpk, while attaching a NIZK proof that the result is a valid ciphertext. For each wire α, we then use the homomorphic properties of the encryption scheme TE to add contributions of all committee members. This ensures that each wire value λ^α is truly random.[2] Given encryptions of these values, we then process the circuit gate by gate. In the following, let c^α denote the ciphertext containing the value λ^α for each wire α. Note that for all values that are published in the following description, we ask the parties to always attach NIZK proofs stating that they did everything correctly. The parties then use for the computation only the values for which the corresponding proofs verify.

For the addition gates, we obtain the encryption of the output wire value by simply adding the ciphertexts containing the encryptions of the input values. For each multiplication gate with

[2] Although it is common in honest majority settings to apply *randomness extraction* to achieve better communication [20], in our case such techniques do not yield any benefits, as, in the YOSO setting, the cost of broadcast is the same as peer-to-peer.

encrypted input wire values c^α, c^β, and encrypted output wire value c^γ, we need to compute the encryption of $\Gamma^\gamma = \lambda^\alpha \cdot \lambda^\beta - \lambda^\gamma$. To do this, we use an encrypted Beaver triple (c^x, c^y, c^z) to compute $c^\epsilon = c^\alpha + c^x$ and $c^\delta = c^\beta + c^y$. Then, we let the current committee use its shares of tsk to decrypt c^ϵ and c^δ. Denote the result by ϵ and δ. Then, we use the homomorphic properties of the encryption scheme to obtain $c^\beta \cdot \epsilon - c^x \cdot \delta + c^z - c^\gamma$, which is precisely the encryption of $\lambda^\beta \cdot (\lambda^\alpha + \lambda^x) - \lambda^x \cdot (\lambda^\beta + \lambda^y) + \lambda^z - \lambda^\gamma = \lambda^\alpha \cdot \lambda^\beta - \lambda^\gamma$.

Finally, in order to ensure that our online phase remains efficient, we pack the random wire values $(\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$ for each group of k input wires $(\alpha_1, \ldots, \alpha_k)$ which belong to a single client. Similarly, we pack the random vector $\boldsymbol{\lambda^\alpha} = (\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$ of the output wires for each group of k multiplication gates. For this we use the fact that we have encryptions $c^{\lambda^{\alpha_1}}, \ldots, c^{\lambda^{\alpha_k}}$: by getting t extra encryptions of random values $c^{r_1}, \ldots, c^{r_t}$ (which can be obtained by letting each committee member prepare t random values $r_{i,j}$, and encrypt each of these values under the global threshold key tpk, and then adding up the contributions), we can use these ciphertexts to homomorphically compute encryptions of the evaluation points of the polynomial $f(x)$ of degree $t + k - 1$, which on each x-point $-(i-1)$ evaluates to λ_i^α, $i \in [k]$, and on i evaluates to r_i, $i \in [t]$. Then, we can use all points (both the values λ^{α_i}, $i \in [k]$ and extra random values r_i, $i \in [1, \ldots, t]$), to locally interpolate using the homomorphic properties of the threshold encryption scheme, and this way obtain *packed* shares $f(1), \ldots, f(n)$ of $\boldsymbol{\lambda^\alpha}$ which are encrypted under tpk.

As the last step, we re-encrypt previously computed packed shares (which are currently encrypted under the tpk) to the KFFs of the roles in the online committee who will use these shares. This step is crucial: If we leave the packed shares encrypted under tpk, then during the online phase one committee will have to spend $O(n)$ communication to decrypt a *single packed share*, with each committee member publishing its CDN-style share of the packed share. This would defy the usage of packing and result in $O(n^2)$ communication per packed sharing, hence linear amortized cost—no better than prior works. Re-encrypting the share towards the KFF of the recipient allows us to pay the linear cost during the offline phase, and keep the online phase very efficient.

For the ease of presentation, we first separately present a few helper functions, which we will use throughout our protocol. We start with $\mathsf{Re\text{-}encrypt}$. This function takes as input a ciphertext c which was encrypted using the threshold key tpk, and enables the current committee C_l to re-encrypt the underlying plaintext using the given key pk. For this, the members of the current committee first recover their shares of tsk. See Figure 1 for details.

Note that the relation R that we prove is the following:

$$R = \left\{ \begin{array}{l} \phi = (pk, tpk, \{pk_j\}_{j\in[n]}, \\ \{c_j^{new}\}_{j\in[n]}, \{c_j^{old}\}_{j\in S}, c^{new}, c^{old}) \\ w = (sk, tsk, r_c, r, \{r_j\}_{j\in[n]}, d, \\ \{s_j^{new}\}_{j\in[n]}, \{s_j^{old}\}_{j\in S}) \end{array} \middle| \begin{array}{l} sk \text{ corresponds to } pk, \\ s_j^{old} \leftarrow \mathsf{PKE.Dec}(sk, c_j^{old}) \text{ for } j \in S, \\ tsk \leftarrow \mathsf{TKRec}(tpk, \{s_j^{old}\}_{j\in S}), \\ \{s_j^{new}\}_{j\in[n]} \leftarrow \mathsf{TRes}(tpk, tsk; r), \\ d \leftarrow \mathsf{TPDec}(tpk, tsk), \\ c^{new} \leftarrow \mathsf{PKE.Enc}(pk, d; r_c), \\ \{c_j^{new}\}_{j\in[n]} \leftarrow \mathsf{PKE.Enc}(pk_j, s_j^{new}; r_j) \\ \text{for } j \in [n] \end{array} \right\}$$

Protocol 1: $\Pi_{\mathsf{YOSO\text{-}Re\text{-}encrypt}}$

Let C_m denote the committee that published encryptions of subshares of tsk that were intended for $C_{l,j}$. Let $\pi_{C_{m,j}}$ denote the corresponding proof by party $C_{m,j}$ that everything was done correctly (details below). Finally, let $c^{\mathsf{TK}}_{C_l,i,j}$ denote the encryption of the i-th subshare of the share of tsk that is intended for $C_{l,j}$.

$\mathsf{Re\text{-}encrypt}_{C_l}(pk, c)$ **:** Each role $C_{l,i}$ of committee C_l does the following:

1. Reconstruct its key share:
   - Let S denote a set of parties in C_m whose proof $\pi_{C_{m,j}}$ verifies.
   - Decrypt encryptions of the subshares of tsk intended for $C_{l,i}$: $s^{old}_{j,i} \leftarrow \mathsf{PKE.Dec}(sk_{C_{l,i}}, c^{\mathsf{TK}}_{C_l,j,i})$ for each $j \in S$.
   - Reconstruct its key share as $tsk_i \leftarrow \mathsf{TRec}(tpk, \{s^{old}_{j,i}\}_{j\in S})$.
2. Compute the partial decryption as $d_i \leftarrow \mathsf{TPDec}(tpk, tsk_i)$.
3. Sample randomness r_{enc} and encrypt the partial decryption under the given public key $c_i \leftarrow \mathsf{PKE.Enc}(pk, d_i; r_{enc})$.
4. Re-share its key share to the committee C_k that will need it next:
   - Sample randomness r and compute $(s^{new}_{i,1}, \ldots, s^{new}_{i,n}) \leftarrow \mathsf{TRes}(tpk, tsk_i; r)$.
   - Sample randomness r_j and compute $c^{\mathsf{TK}}_{C_k,i,j} \leftarrow \mathsf{PKE.Enc}(pk_{C_{k,j}}, s_{i,j}; r_j)$ for each $j \in [n]$.
   - Compute a NIZK proof $\pi_{C_l,i}$ that everything was done correctly:

$$\pi_{C_{l,i}} \leftarrow \mathsf{NIZKAoK.P}\left(\begin{array}{l} crs, \\ \phi = (pk_{C_{l,i}}, tpk, \{pk_{C_{k,j}}\}_{j\in[n]}, \{c^{\mathsf{TK}}_{C_k,i,j}\}_{j\in[n]}, \\ \{c^{\mathsf{TK}}_{C_l,j,i}\}_{j\in S}, c_i, c), \\ w = (sk_{C_{l,i}}, tsk_i, r_{enc}, r, \{r_j\}_{j\in[n]}, d, \\ \{s^{new}_{i,j}\}_{j\in[n]}, \{s^{old}_{j,i}\}_{j\in S}) \end{array}\right)$$

5. Broadcast $(c^{\mathsf{TK}}_{C_k,i,1}, \ldots, c^{\mathsf{TK}}_{i,n})$ along with $\pi_{C_l,i}$.
6. Broadcast c_i.

We further use the function $\mathsf{Decrypt}$, which is essentially the same as $\mathsf{Re\text{-}encrypt}$, except that instead of re-encrypting the obtained share under a new public key, we simply broadcast it in clear. See Figure 3 for details, where the NIZK relation that we prove is the following:

$$R = \left\{ \begin{array}{l} \phi = (pk, tpk, \{pk_j\}_{j\in[n]}, \\ \{c^{new}_j\}_{j\in[n]}, \{c^{old}_j\}_{j\in S}, c^{old}) \\ w = (sk, tsk, r_c, r, \{r_j\}_{j\in[n]}, d, \\ \{s^{new}_j\}_{j\in[n]}, \{s^{old}_j\}_{j\in S}) \end{array} \middle| \begin{array}{l} sk \text{ corresponds to } pk, \\ s^{old}_j \leftarrow \mathsf{PKE.Dec}(sk, c^{old}_j) \text{ for } j \in S, \\ tsk \leftarrow \mathsf{TKRec}(tpk, \{s^{old}_j\}_{j\in S}), \\ \{s^{new}_j\}_{j\in[n]} \leftarrow \mathsf{TRes}(tpk, tsk; r), \\ d \leftarrow \mathsf{TPDec}(tpk, tsk), \\ \{c^{new}_j\}_{j\in[n]} \leftarrow \mathsf{PKE.Enc}(pk_j, s^{new}_j; r_j) \\ \text{for } j \in [n] \end{array} \right\}$$

Protocol 2: $\Pi_{\mathsf{YOSO\text{-}Decrypt}}$

Let C_m denote the committee that published encryptions of subshares of tsk that were intended for $C_{l,j}$. Let $\pi_{C_{m,j}}$ denote the corresponding proof by party $C_{m,j}$ that everything was done correctly (details below). Finally, let $c^{\mathsf{TK}}_{C_l,i,j}$ denote the encryption of the i-th subshare of the share of tsk that is intended for $C_{l,j}$.

$\mathsf{Decrypt}_{C_l}(c)$ **:** Each role $C_{l,i}$ of committee C_l does the following:

1. Reconstruct its key share:
   - Let S denote a set of parties in C_m whose proof $\pi_{C_{m,j}}$ verifies.

- Decrypt encryptions of the subshares of tsk intended for $C_{l,i}$: $s_{j,i}^{old} \leftarrow \mathsf{PKE.Dec}(sk_{C_{l,i}}, c_{C_l,j,i}^{\mathsf{TK}})$ for each $j \in S$.
- Reconstruct its key share as $tsk_i \leftarrow \mathsf{TRec}(tpk, \{s_{j,i}^{old}\}_{j\in S})$.

2. Compute the partial decryption as $d_i \leftarrow \mathsf{TPDec}(tpk, tsk_i)$.
3. Re-share its key share to the committee C_k that will need it next:
   - Sample randomness r and compute $(s_{i,1}^{new}, \ldots, s_{i,n}^{new}) \leftarrow \mathsf{TRes}(tpk, tsk_i; r)$.
   - Sample randomness r_j and compute $c_{C_k,i,j}^{\mathsf{TK}} \leftarrow \mathsf{PKE.Enc}(pk_{C_{k,j}}, s_{i,j}; r_j)$ for each $j \in [n]$.
   - Compute a NIZK proof $\pi_{C_l,i}$ that everything was done correctly:

$$\pi_{C_{l,i}} \leftarrow \mathsf{NIZKAoK.P}\left(\begin{array}{l} crs, \\ \phi = (pk_{C_{l,i}}, tpk, \{pk_{C_{k,j}}\}_{j\in[n]}, \{c_{C_k,i,j}^{\mathsf{TK}}\}_{j\in[n]}, \\ \{c_{C_l,j,i}^{\mathsf{TK}}\}_{j\in S}, c), \\ w = (sk_{C_{l,i}}, tsk_i, r_{enc}, r, \{r_j\}_{j\in[n]}, d, \\ \{s_{i,j}^{new}\}_{j\in[n]}, \{s_{j,i}^{old}\}_{j\in S}) \end{array}\right)$$

4. Broadcast $(c_{C_k,i,1}^{\mathsf{TK}}, \ldots, c_{i,n}^{\mathsf{TK}})$ along with $\pi_{C_l,i}$.
5. Broadcast d_i.

Finally, we use the following function to have two committees C_1 and C_2 prepare a Beaver triples. The relation R that the members of the committee C_2 have to prove is the following:

$$R = \left\{ \begin{array}{l|l} \phi = (tpk, c^a, c_i^b, c_i^c) & c_i^b \leftarrow \mathsf{TEnc}(tpk, b_i; r_i), \\ w = (b, r) & c_i^c \leftarrow \mathsf{TEval}(tpk, c^a, b_i) \end{array} \right\}$$

Protocol 3: $\Pi_{\mathsf{YOSO\text{-}Beaver\text{-}Triples}}$

Let C_1 and C_2 denote two committees that are participating in the generation of the Beaver triple. $\mathsf{Beaver\text{-}Triple}_{\boldsymbol{C_1},\boldsymbol{C_2}}$ **:**

- Each role $C_{1,i}$ of committee C_1 does the following:
  - Generate a random value a_i.
  - Generate randomness r_i, and encrypt a_i via $c_i^a \leftarrow \mathsf{TEnc}(tpk, a_{i,j}; r_i)$.
  - Broadcast c_i^a along with a NIZK proof $\pi_{C_1,i}$ that the ciphertext was computed correctly.
- Let S denote a set of roles in C_1 whose proof $\pi_{C_1,i}$ verifies. Let $|S|$ denote the size of S.
- Everyone can now compute $c^a \leftarrow \mathsf{TEval}(tpk, \{c_i^a\}_{i\in S}, (1)^{|S|})$.
- Each role $C_{2,i}$ of committee C_2 does the following:
  - Generate a random value b_i.
  - Generate randomness r_i, and encrypt b_i via $c_i^b \leftarrow \mathsf{TEnc}(tpk, b_i; r_i)$.
  - Compute $c_i^c \leftarrow \mathsf{TEval}(tpk, c^a, b_i)$
  - Compute a NIZK proof $\pi_{C_2,i}$ that everything was done correctly:

$$\pi_{C_{l,i}} \leftarrow \mathsf{NIZKAoK.P}\left(\begin{array}{l} crs, \\ \phi = (tpk, c^a, c_i^b, c_i^c), \\ w = (b_i, r_i) \end{array}\right)$$

  - Broadcast b_i, c_i along with the proof $\pi_{C_2,i}$.

- Let S' denote a set of roles in C_2 whose proof $\pi_{C_2,i}$ verifies. Let $|S'|$ denote the size of S'.
- Everyone can now compute $c^a \leftarrow \mathsf{TEval}(tpk, \{c_i^b\}_{i\in S'}, (1)^{|S'|})$ and $c^c \leftarrow \mathsf{TEval}(tpk, \{c_i^c\}_{i\in S'}, (1)^{|S'|})$
- The resulting triple is (c^a, c^b, c^c).

We now give the full description of the offline phase in Figure 4.

Protocol 4: $\Pi_{\mathsf{YOSO\text{-}Offline}}$

Offline phase:
Let C denote the circuit.
Step 1: Prepare Beaver Triples Let the first two committees C_1^{Off} and C_2^{Off} prepare a Beaver triple for each multiplication gate of the circuit via $\mathsf{Beaver\text{-}Triple}_{C_1,C_2}$.
Step 2: Prepare random wire values
For each wire α of C that is an output wire of an input/multiplication gate:

1. Each $C_{3,i}^{\mathsf{Off}}$ does the following:
   - Generate a random value λ_i^α.
   - Encrypt λ_i^α via $c_i^\alpha \leftarrow \mathsf{TEnc}(tpk, \lambda_i^\alpha)$.
   - Broadcast c_i^α along with a NIZK proof $\pi_{C_{3,i}^{\mathsf{Off}}}$ that the ciphertext was computed correctly.
2. Let S denote the set of roles $C_{3,i}^{\mathsf{Off}}$ whose proofs verified correctly.
3. Everyone computes $c^\alpha \leftarrow \mathsf{TEval}(tpk, \{c_i^\alpha\}_{i\in S}, (1)^{|S|})$.

Step 3: Compute dependent wire values
For each *addition gate* with inputs wires α and β, and output wire γ, set (from lowest depth gates to highest):

- $c^\gamma \leftarrow \mathsf{TEval}(tpk, (c^\alpha, c^\beta), (1, 1))$.

For each *multiplication gate* with inputs wires α and β, and output wire γ, compute the encryption c^{Γ^γ} of the value $\lambda^\alpha * \lambda^\beta - \lambda^\gamma$ on wire γ. For this, let (c^x, c^y, c^z) denote an unused Beaver triple (from the ones prepared in Step 1). Then, compute the following for groups of k multiplication gates in parallel:

- Everyone computes $c^\epsilon \leftarrow \mathsf{TEval}(tpk, (c^\alpha, c^x), (1, 1))$ and $c^\delta \leftarrow \mathsf{TEval}(tpk, (c^\beta, c^y), (1, 1))$.
- Current committee C_l^{Off} decrypts $\epsilon = \mathsf{Decrypt}_{C_l^{\mathsf{Off}}}(c^\epsilon)$ and $\delta = \mathsf{Decrypt}_{C_l^{\mathsf{Prep}}}(c^\delta)$.
- Everyone computes $c^{\Gamma^\gamma} \leftarrow \mathsf{TEval}(tpk, (c^\beta, c^x, c^z, c^\gamma), (\epsilon, -\delta, 1, -1))$.

Step 4: Pack values for multiplication gates
For each group of k multiplication gates, let $(c^{\alpha_1}, \ldots, c^{\alpha_k})$ denote the ciphertexts containing the values on the corresponding output wires. Each role $C_{l,i}^{\mathsf{Off}}$ of the current committee C_l^{Off} does the following:

- Generate t random values $(r_i^{k+1}, \ldots, r_i^{t+k})$.
- Let $c_i^{\mathsf{Help},\alpha_j} \leftarrow \mathsf{TEnc}(tpk, r_i^j)$ for $j \in [k+1, \ldots, t+k]$
- Broadcast $\{c_i^{\mathsf{Help},\alpha_j}\}_{j\in[k+1,\ldots,t+k]}$ along with a NIZK proof $\pi_{C_{l,i}^{\mathsf{Off}}}$ that the ciphertext was computed correctly.

Now, everyone does the following:

- Let S denote the set of roles $C^{\mathsf{Off}}_{l,i}$ whose proofs verified correctly.
- Compute $c^{\mathsf{Help},\alpha_j} \leftarrow \mathsf{TEval}(tpk, \{c_i^{\mathsf{Help},\alpha_j}\}_{i\in S}, (1)^{|S|})$ for each $j \in [k+1, t+k]$.
- For each $i \in [n]$, compute the share of the i-th committee role as:

$$c_i^{\boldsymbol{\alpha}} \leftarrow \mathsf{TEval}(tpk, (c^{\alpha_1}, \ldots, c^{\alpha_k}, c^{\mathsf{Help},\alpha_{k+1}}, \ldots, c^{\mathsf{Help},\alpha_{t+k}}), (l_1(i), \ldots, l_{t+k}(i))),$$

  where $l_j(\cdot)$ denotes the j-th Lagrange basis polynomial.

Similarly, for each group of k multiplication gates, pack the ciphertexts containing c^{Γ^γ}.

Step 5: Prepare ciphertexts to the future, input gates

For each input gate α that belongs to the i-th client, let c^α denote the ciphertext containing the value on the corresponding output wire. Current committee C^{Off}_l computes the following for all $i \in [n]$ in parallel:

- $c^\alpha_{P_i} \leftarrow \mathsf{Re\text{-}encrypt}_{C^{\mathsf{Off}}_l}(pk^{\mathsf{KFF}}_{P_i}, c^\alpha)$

Step 6: Prepare ciphertexts to the future, multiplication gates

For each group of k multiplication gates, let $c_i^{\boldsymbol{\alpha}}$ denote the encryption of the i-th (packed) share of the left input wires, $c_i^{\boldsymbol{\beta}}$ the encryption of the i-th (packed) share of the right input wires, and $c_i^{\boldsymbol{\Gamma^\gamma}}$ denote the encryption of i-th (packed) share of the values $\lambda_{\boldsymbol{\alpha}} * \lambda_{\boldsymbol{\beta}} - \lambda_{\boldsymbol{\gamma}}$ for the vector of corresponding output wires $\boldsymbol{\gamma}$. Let C_m denote the committee that will be evaluating gates γ in the online phase. Current committee C^{off}_l computes the following for all $i \in [n]$ in parallel:

- $c^{\boldsymbol{\alpha}}_{C_{m,i}} \leftarrow \mathsf{Re\text{-}encrypt}_{C^{\mathsf{off}}_l}(pk^{\mathsf{KFF}}_{C_{m,i}}, c_i^{\boldsymbol{\alpha}})$
- $c^{\boldsymbol{\beta}}_{C_{m,i}} \leftarrow \mathsf{Re\text{-}encrypt}_{C^{\mathsf{off}}_l}(pk^{\mathsf{KFF}}_{C_{m,i}}, c_i^{\boldsymbol{\beta}})$
- $c^{\boldsymbol{\Gamma^\gamma}}_{C_{m,i}} \leftarrow \mathsf{Re\text{-}encrypt}_{C^{\mathsf{Off}}_l}(pk^{\mathsf{KFF}}_{C_{m,i}}, c_i^{\boldsymbol{\Gamma^\gamma}})$

Communication analysis. In Step 1, we need $O(n)$ communication to prepare a Beaver triple [3]. We require $O(n)$ communication to prepare each random wire value in Step 2. Next, Step 3 requires $\frac{O(n^2)}{k} = O(n)$ communication per gate. In Step 4, the communication cost is again $\frac{O(n^2)}{k} = O(n)$ per gate. Re-encrypting values in Steps 5 and 6 each requires $O(n)$ communication per re-encrypted value, i.e., per wire, plus $O(n^2)$ one-time cost. In total, the communication complexity of our offline phase is $O(n|C|)$.

5.3 Online phase

During the online phase, our goal is to compute values $\mu_\alpha = v_\alpha - \lambda_\alpha$ for each wire of the circuit, where v_α is the actual wire value, and λ_α the random value prepared for this wire during the offline phase. To do this, we must first let the roles of the online committees obtain the random values prepared for these roles during the preprocessing. Note that during the offline phase, we were encrypting the secret random values under the KFFs of the corresponding roles' of the online phase. Hence, it is sufficient to let the online roles learn the secret key portion of their corresponding KFF. For this, we have parties of the first online committee use their shares of tsk to re-encrypt the secret portions of the KFFs towards the *role keys of the YOSO role-generation*, which can now safely assumed to be known.

[3] Note that our offline phase is bottlenecked by a step which requires linear communication, hence for simplicity we skip potential optimizations that could amortize the cost of non-bottleneck steps to a constant one.

For the input gates, given input v_α for the input wire α, each client can simply use its secret KFF to decrypt the value λ_α and broadcast $v_\alpha - \lambda_\alpha$.

For the addition gates, given input wires α and β, by invariant we know both μ_α and μ_β. Hence, everyone can locally compute $\mu_\alpha + \mu_\beta$.

For the multiplication gates, given input wire vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, and output wire vector $\boldsymbol{\gamma}$, the i-th member of the current committee can use the packed shares of $\lambda_i^{\boldsymbol{\Gamma}^\gamma}$ which it has from the offline phase, along with the publicly reconstructed $\boldsymbol{\mu_\alpha}$ and $\boldsymbol{\mu_\beta}$, to obtain $\mu_i^{\boldsymbol{\gamma}} = \mu_i^{\boldsymbol{\alpha}} * \mu_i^{\boldsymbol{\beta}} + \mu_i^{\boldsymbol{\alpha}} * \lambda_i^{\boldsymbol{\beta}} + \mu_i^{\boldsymbol{\beta}} * \lambda_i^{\boldsymbol{\alpha}} + \lambda_i^{\boldsymbol{\Gamma}^\gamma}$.

Finally, for each output wire α, we let the last committee use its shares of tsk to re-ecrypt the preprocessed value λ^α towards the public key of the client P_i, who is supposed to get the output of this wire. Then, P_i can use the publicly available μ^α to reconstruct v^α as $\mu^\alpha + \lambda^\alpha$.

Protocol 5: $\Pi_{\mathsf{YOSO-Online}}$

Let pk_{P_i} denote public key of the client P_i, and $pk_{C_{l,i}}$ denote YOSO role-assignment key of the committee member $C_{l,i}$.

Online phase:

Future key distribution

The first online committee C_1 computes the following for all clients and all future committee roles $C_l[i]$ in parallel:

- $c_{P_i}^{\mathsf{KFF'}} \leftarrow \mathsf{Re\text{-}encrypt}_{C_1}(pk_{P_i}, c_{P_i}^{\mathsf{KFF}})$
- $c_{C_{l,i}}^{\mathsf{KFF'}} \leftarrow \mathsf{Re\text{-}encrypt}_{C_1}(pk_{C_{l,i}}, c_{C_{l,i}}^{\mathsf{KFF}})$

Input

For each input gate that belong client P_i, let $c_{P_i}^{\alpha}$ denote the ciphertext containing the prepared random value on the corresponding output wires. Each client P_i uses its secret role key sk_{P_i} from the YOSO role-assignment to compute the following:

- Let $sk_{P_i}^{\mathsf{KFF}} \leftarrow \mathsf{Dec}(sk_{P_i}, c_{P_i}^{\mathsf{KFF'}})$
- Let $\lambda^\alpha \leftarrow \mathsf{Dec}(sk_{P_i}^{\mathsf{KFF}}, c_{P_i}^{\alpha})$
- Compute and publish $\mu^\alpha = v^\alpha - \lambda^\alpha$

Addition

For two addition gates with input wires α and β, everyone can locally compute $\mu^\alpha + \mu^\beta$.

Multiplication

Let C_l denote the current committee. For each group of multiplication gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, let $c_{C_{l,i}}^{\boldsymbol{\alpha}}, c_{C_{l,i}}^{\boldsymbol{\beta}}, c_{C_{l,i}}^{\boldsymbol{\Gamma}^\gamma}$ denote the ciphertexts containing the share prepared for the i-th role of C_l. Let $\boldsymbol{\mu_\alpha}$ and $\boldsymbol{\mu_\beta}$ denote the publicly reconstructed values on wires $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. Let $\mu_i^{\boldsymbol{\alpha}}$ and $\mu_i^{\boldsymbol{\beta}}$ denote the i-th share of a degree-$(k-1)$ packed sharing of each of this vectors. Each member of the current committee uses its secret role key $sk_{C_{l,i}}$ from the YOSO role-assignment to compute the following:

- Let $sk_{C_{l,i}}^{\mathsf{KFF}} \leftarrow \mathsf{Dec}(sk_{C_{l,i}}, c_{C_{l,i}}^{\mathsf{KFF'}})$
- Let $\lambda_i^{\boldsymbol{\alpha}} \leftarrow \mathsf{Dec}(sk_{C_{l,i}}^{\mathsf{KFF}}, c_{C_{l,i}}^{\boldsymbol{\alpha}})$
- Let $\lambda_i^{\boldsymbol{\beta}} \leftarrow \mathsf{Dec}(sk_{C_{l,i}}^{\mathsf{KFF}}, c_{C_{l,i}}^{\boldsymbol{\beta}})$
- Let $\lambda_i^{\boldsymbol{\Gamma}^\gamma} \leftarrow \mathsf{Dec}(sk_{C_{l,i}}^{\mathsf{KFF}}, c_{C_{l,i}}^{\boldsymbol{\Gamma}^\gamma})$
- Compute and publish $\mu_i^{\boldsymbol{\gamma}} = \mu_i^{\boldsymbol{\alpha}} * \mu_i^{\boldsymbol{\beta}} + \mu_i^{\boldsymbol{\alpha}} * \lambda_i^{\boldsymbol{\beta}} + \mu_i^{\boldsymbol{\beta}} * \lambda_i^{\boldsymbol{\alpha}} + \lambda_i^{\boldsymbol{\Gamma}^\gamma}$ along with the proof of correctness.

In order to reconstruct $\boldsymbol{\mu}^\gamma$, anyone can use $t + 2(k-1) + 1$ shares $\boldsymbol{\mu}_i^\gamma$ that verified correctly.
Output
The last committee C_l uses its shares of tsk to re-encrypt (in parallel) each value c^α towards the client P_i who is obtaining the output of wire α.

- $c^\alpha_{P_i} \leftarrow \mathsf{Re\text{-}encrypt}^*_{C_1}(pk_{P_i}, c^\alpha)$

Above, Re-encrypt* is the same as Re-encrypt, except that we do not distribute shares of tsk anymore. This reduces the overall communication per output wire to $O(n)$.
To obtain the output for a wire α, client P_i uses sk_{P_i} to decrypt the ciphertext $c^\alpha_{P_i}$ containing the corresponding random wire value. Then, the client computes the output using the public value μ_α:

- Let $\lambda^\alpha \leftarrow \mathsf{Dec}(sk^{\mathsf{KFF}}_{P_i}, c^\alpha_{P_i})$.
- Compute $v^\alpha = \mu^\alpha + \lambda^\alpha$.

Communication analysis. The future key distribution step results in $O(n)$ communication per role, as each committee member must publish its share of the value that is being re-encrypted, plus a one-time cost of $O(n^2)$ for the re-distribution of the shares of tsk. Further, the input step requires a one-time cost that is linear in the number of inputs. Addition gates do not require any communication, and a batch of $O(n)$ multiplication gates requires each committee member to publish its share of $\boldsymbol{\mu}^\gamma$ along with a proof of correctness, and thus incurs $O(n)$ cost per batch. Finally, the output layer requires $O(n)$ communication per output wire. Assuming that each role processes $O(n)$ values, this gives us $O(1)$ amortized communication per gate.

This construction allows us to obtain the following result:

Theorem 1. *Assuming a secure broadcast and role-assignment, for any n-party function F, protocol $\Pi = (\Pi_{\mathsf{YOSO\text{-}Setup}}, \Pi_{\mathsf{YOSO\text{-}Offline}}, \Pi_{\mathsf{YOSO-Online}})$, YOSO-securely implements the ideal functionality $\mathcal{F}^F_{MPC}$ for a corruption threshold $t < \frac{n}{2} \cdot (1-\epsilon)$. The offline communication complexity is $O(n)$ elements per gate, the online communication complexity is $O(1)$ elements per gate.*

We defer the formal proof to Section B in the Supplementary Material.

5.4 Bonus: Supporting Fail-Stop Parties

We observe that, in our protocol, the online phase requires at least $t+2(k-1)+1$ partial decriptions to be posted. There are $n-t$ honest parties, so we need to ensure that $n - t \geq t + 2(k-1) + 1$ for GOD, or $n > 2t + 2(k-1)$. Assuming $t < n(\frac{1}{2} - \epsilon)$, this is equivalent to $n \geq n(1-2\epsilon) + 2(k-1)$, or $k - 1 \leq n\epsilon$ This allows us to get a saving factor in the online phase of $k \approx n\epsilon$.

However, note we can do the following. Write $\epsilon = 2\epsilon'$, and set $k = n\epsilon' + 1$. We have that $t + 2(k-1) + 1 < n(\frac{1}{2} - \epsilon) + 2(n\epsilon') + 1 = n/2 + 1$. On the other hand, the amount of honest parties is $n - t > n - n(\frac{1}{2} - \epsilon) = n(\frac{1}{2} + \epsilon)$, which is at least $n\epsilon$ more than the required amount of parties for reconstruction. We conclude the following: by reducing the packing parameter from $\approx n\epsilon$ to $\approx n\epsilon/2$, the protocol is able to proceed, even if $n \cdot \epsilon$ *honest parties* do not participate. We believe this property is crucial for YOSO MPC protocols, which are intended to be deployed in settings with large number of parties and hence are prone to non-malicious failures such as slowdowns, hardware/software errors, and others.

6 Role Assignment: Committee Size Analysis

In previous sections we have described a YOSO MPC protocol with high efficiency, assuming that the committee sizes n are such that the amount of corrupted parties in each committee is upper bounded by $t < n(\frac{1}{2} - \epsilon)$. Ensuring this corruption threshold is in charge of the role assignment layer, which is generally a separate task to that of YOSO MPC. Different works have proposed different role assignment layers for YOSO [6], but generally, these approaches focus on obtaining committees with 1/2 corruption ratios. In this section we make use of the analysis from [6]—which is itself similar to the one from [28]—in order to obtain the stronger honest majority bound $t < n(\frac{1}{2} - \epsilon)$. Briefly recall that Benhamouda *et al.*'s role assignment works as follows: to select members of the MPC committee, a so-called *nominating committee* is first formed by self-selection via a process known as *cryptographic sortition*. Each nominator then chooses a member of the MPC committee, and generates short-term public/secret key pair $(\mathsf{epk}, \mathsf{esk})$. The nominator encrypts esk under the chosen party's long-term key to obtain a ciphertext c, and publishes the pair (epk, c). Finally, each party can attempt to decrypt the ciphertexts c to check whether they were selected to participant in the MPC committee or not. We refer the reader to [6] for details.

For this section, we stick to the notation from [6, Section 3.2], so that we can easily reuse their analysis. Let N be the total number of parties, among which committees will be sampled, and let us assume that the adversary corrupts $f \cdot N$ out of these N parties. Cryptographic sortition is a *probabilistic* process that samples committees by including each party in the committee with some probability C/N.[4] Let c be the random variable denoting the size of the committee, and let ϕ be the random variable denoting the number of corrupted parties in the selected committee. Our goal is to determine a threshold t so that, with high probability, $\phi < t$ and $t \leq c \cdot (\frac{1}{2} - \epsilon)$. Note that t in [6] has a different connotation than in our previous sections: here, $t - 1$ corresponds an upper bound (with high probability) for the number of corruptions in a committee. This is roughly equivalent to what we refer to as "t" in prior sections.

Let k_1, k_2 and k_3 be three security parameters for the analysis, as follows.

1. The adversary can try to win the cryptographic sortition at most 2^{k_1} times
2. We want to ensure that $\phi < t$ with probability $\geq 1 - 2^{k_2}$
3. We want to ensure that $t \leq c \cdot (\frac{1}{2} - \epsilon)$ with probability $\geq 1 - 2^{k_3}$.

In [6], the threshold t is written as $t = B_1 + B_2 + 1$, where $B_1 = fC(1 + \epsilon_1)$ and $B_2 = f(1 - f)C(1 + \epsilon_2)$, for some $\epsilon_1, \epsilon_2 > 0$. In [6] it is shown that, if

$$C > \max\left\{ \frac{(k_1 + k_2 + 1)(2 + \epsilon_1)\ln 2}{f\epsilon_1^2},\ \frac{(k_2 + 1)(2 + \epsilon_2)\ln 2}{f(1 - f)\epsilon_2^2} \right\}, \tag{2}$$

then Item 2 from above holds, that is, the number of corruptions ϕ among the elected committee is at most t, except with probability 2^{-k_2}.

Regarding Item 3, we note that $t < c \cdot (\frac{1}{2} - \epsilon)$ is equivalent to the number of honest parties $c - t$ being greater than $\frac{1/2+\epsilon}{1/2-\epsilon} \cdot t$. In [6] this is shown to hold with probability $\geq 1 - 2^{k_3}$ if, for some $\epsilon_3 > 0$:

$$C > \max\left\{ \frac{2k_3 \ln 2}{(\epsilon_3(1 - f))^2},\ \frac{(\frac{1}{2} + \epsilon) \cdot (fC(1 + \epsilon_1) + f(1 - f)C(1 + \epsilon_2))}{(\frac{1}{2} - \epsilon)(1 - f)^2(1 - \epsilon_3)} \right\} \tag{3}$$

[4] Note that, even though the *expected size* of the committee is C, its concrete size is variable.

Interestingly, this is a generalization of [6], with their set of constraints being achieved by setting $\epsilon = 0$. We see then that the only difference with respect to their work is the factor $\frac{1/2+\epsilon}{1/2-\epsilon}$ in Eq. (3). That is, the cost to pay to enable the gap $\epsilon > 0$ is reflected in this term.

In what follows, let us fix as in [6] $k_1 = 64$, $k_2 = k_3 = 128$. Set ϵ_1 and ϵ_2 as small as possible so that Eq. (2) holds. Using numerical methods we find this is

$$\epsilon_1 > \frac{1}{2}\sqrt{\frac{1544Cf\ln(2) + 37249\ln^2(2)}{C^2f^2}} + \frac{193\ln(2)}{2Cf} \tag{4}$$

and

$$\epsilon_2 > \frac{1}{2}\sqrt{\frac{-1032Cf^2\ln(2) + 1032Cf\ln(2) + 16641\ln^2(2)}{C^2(f^2-f)^2}} - \frac{129\ln(2)}{2C(f^2-f)}. \tag{5}$$

Note these values—which determine t—only depend on C and f. Now, let ϵ_3 to be the smallest value that satisfies (3), we have:

$$\sqrt{\frac{256\ln 2}{C(1-f)^2}} < \epsilon_3 < 1 - \left(\frac{\frac{1}{2}+\epsilon}{\frac{1}{2}-\epsilon}\right) \cdot \frac{(fC(1+\epsilon_1) + f(1-f)C(1+\epsilon_2))}{(1-f)^2C} \tag{6}$$

Let us denote $\delta = \frac{\frac{1}{2}+\epsilon}{\frac{1}{2}-\epsilon}$, which satisfies $1 \leq \delta$ for $0 < \epsilon < 1/2$. Note that the most efficient choice is to take ϵ_1 and ϵ_2 as small as possible, according to Eqs. (4) and (5). Then, set ϵ_3 as small as possible according to Eq. (6). At this point, $\delta > 1$ must satisfy the right inequality of Eq. (6). The work of [6] took $\delta = 1$, for which $\epsilon = 0$. Here, we note that we can take some $\delta > 1$, which corresponds to $\epsilon > 0$, as long as the inequality still holds.

In Table 1 we present a selection of parameters obtained with the reasoning above. We choose the sortition parameter C in $\{1000, 5000, 10000, 20000, 40000\}$ (which is the *expected* size of the committee), and the global corruption ratio f in $\{0.05, 0.10, 0.15, 0.20, 0.25\}$. Then, we compute $\epsilon_1, \epsilon_2, \epsilon_3$ as described above. This determines t, where $t-1$ is (w.h.p.) an upper bound the number of corruptions in the committee, and along with it $c = t/(1/2-\epsilon)$ is determined, which is (w.h.p.) a lower bound on the size of the committee. Note that the lower bound that is guaranteed (w.h.p.) in [6] is $c' = 2t$. However, our analysis shows that the committee size is actually larger than that, and that gap can be used to gain a $k = n \cdot \epsilon$ improvement factor. Overall, we see that by increasing the committee size from c' to c, which as f grows this becomes a less pronounced jump, we can actually reduce communication by a factor of k, which can be as big as three orders of magnitude! For instance, setting $C = 20000$, for 20% global corruptions, we can take a committee of size $\approx$ 20k instead of $\approx$ 18k from [6] and get an improvement factor in terms of online communication complexity of $> 1000\times$. For smaller f this factor improves even more, at the expense of having a bigger difference in committee sizes. This disparity can be reduced by settling for a smaller packing factor.

7 Conclusions and Future Work

We have shown that YOSO MPC can benefit greatly from transitioning from a setting where $t < n/2$, to $t < n(1/2-\epsilon)$. In large-scale scenarios such as YOSO it is reasonable to assume the adversary corrupts strictly less than a minority, and a "gap" can be exploited in order to get concrete benefits in terms of online communication, as well as fail-stop tolerance. Our work shows how packed secret-sharing can be used to materialize such communication savings. Furthermore, we have shown that requiring such gap does not substantially affect the size of elected committees.

C	f	t	c	c'	ϵ	k
1000	0.05	446	949	893	0.03	28
	0.1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
	0.15	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
	0.2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
	0.25	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
5000	0.05	1078	4699	2157	0.27	1271
	0.1	1721	4925	3444	0.15	741
	0.15	2293	5106	4588	0.05	259
	0.2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
	0.25	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
10000	0.05	1754	9518	3509	0.32	3004
	0.1	2937	9841	5876	0.20	1982
	0.15	4004	10098	8009	0.10	1045
	0.2	4983	10319	9968	0.02	175
	0.25	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
20000	0.05	2998	19264	5998	0.34	6633
	0.1	5216	19723	10433	0.24	4645
	0.15	7237	20088	14476	0.14	2806
	0.2	9107	20401	18215	0.05	1093
	0.25	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
40000	0.05	5331	38907	10664	0.36	14121
	0.1	9552	39558	19106	0.26	10226
	0.15	13437	40074	26875	0.16	6600
	0.2	17047	40517	34096	0.08	3211
	0.25	20408	40911	40818	0.01	47

Table 1: Sample parameters. C is the parameter for the cryptographic sortition, that is, each party from the global pool is chosen with prob. C/N. f is the global corruption ratio, that is, there are $f \cdot N$ corrupt parties among the global N parties. t is the threshold that upper-bounds the number of corruptions (plus one) in the committee (w.h.p.). $c = t/(\frac{1}{2} - \epsilon)$ is the lower bound on the committee size (w.h.p.), and $c' = 2t$ is the lower bound on the committee size if one takes $\epsilon = 0$ (w.h.p.). ϵ is the gap. k is the packing factor. $\perp$ means that the given ratio f is impossible for the given value of C.

Our work serves as a starting point to fully unleash the practicality of YOSO MPC. Relevant follow-up works include the following:

- Instantiating our framework with class groups-based solutions such as [14, 10], which remove the trusted setup. We did not follow this approach in our work since (1) it is simpler in terms of exposition to consider the original presentation from [29], and (2) class group-based solutions have a $n!$ overhead in terms of communication due to the use of integer secret-sharing, which may not be suitable for large-party settings.
- We adapted the role assignment from [6], showing that demanding for a gap does not affect committee sizes drastically (specially given the benefits). It remains to be seen what is the impact of this condition in more recent role assignment protocols such as [11].
- As a feasibility result, it is interesting to explore what the impact of the "gap" is in the context of the *information-theoretic security*, where no computational assumptions are used at the protocol level.
- Our preprocessing unfortunately does not benefit from the packing parameter k. This is an inherent limitation of Turbopack[25], and we find it highly relevant to remove such limitation.

Acknowledgments

This paper was prepared in part for information purposes 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. 2025 JP Morgan Chase & Co.

References

[1] Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. SCALES - MPC with small clients and larger ephemeral servers. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography*, 2022.

[2] Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. Malicious security for SCALES - outsourced computation with ephemeral servers. In *Annual International Cryptology Conference*, 2024.

[3] Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure MPC: Laziness leads to GOD. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 120–150. Springer, Heidelberg, December 2020.

[4] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

[5] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, 1988.

[6] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 260–290. Springer, Heidelberg, November 2020.

[7] 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.

[8] 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, Heidelberg, August 2023.

[9] Gabriel Bracha. An o (log n) expected rounds randomized byzantine generals protocol. *Journal of the ACM (JACM)*, 34(4):910–920, 1987.

[10] Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure multiparty computation from threshold encryption based on class groups. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 613–645. Springer, Heidelberg, August 2023.

[11] Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future - A paradigm for sending secret messages to future (anonymous) committees. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 151–180. Springer, Heidelberg, December 2022.

[12] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS*, 2001.

[13] Ran Canetti, Sebastian Kolby, Divya Ravi, Eduardo Soria-Vazquez, and Sophia Yakoubov. Taming adaptivity in YOSO protocols: The modular way. In *Theory of Cryptography*, 2023.

[14] Ignacio Cascudo and Bernardo David. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2024.

[15] 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, Heidelberg, December 2022.

[16] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[17] 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, Heidelberg.

[18] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology - EUROCRYPT, International Conference on the Theory and Application of Cryptographic Techniques*, 2001.

[19] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *Int. J. Inf. Sec.*, 9(6):371–385, 2010.

[20] 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, Heidelberg, August 2007.

[21] Ivan Damgård, Daniel Escudero, and Antigoni Polychroniadou. Phoenix: Secure Computation in an Unstable Network with Dropouts and Comebacks. In Kai-Min Chung, editor, *4th Conference on Information-Theoretic Cryptography (ITC 2023)*, volume 267 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:21, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[22] 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, Heidelberg, August 2023.

[23] Giovanni Deligios, Aarushi Goel, and Chen-Da Liu-Zhang. Maximally-fluid MPC with guaranteed output delivery. *IACR Cryptol. ePrint Arch.*, 2023.

[24] Giovanni Deligios, Anders Konring, Chen-Da Liu-Zhang, and Varun Narayanan. Statistical layered mpc. In *Theory of Cryptography Conference, TCC*, 2024.

[25] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. TurboPack: Honest majority MPC with constant online communication. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 951–964. ACM Press, November 2022.

[26] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 121–136. Springer, Heidelberg, August 1998.

[27] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 1992.

[28] 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, Heidelberg, August 2017.

[29] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. 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, Heidelberg.

[30] Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. Random-index PIR and applications. In *Theory of Cryptography - 19th International Conference, TCC*, 2021.

[31] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In *International Conference on Practice and Theory of Public-Key Cryptography*, 2022.

[32] Vipul Goyal, Elisaweta Masserova, Bryan Parno, and Yifan Song. Blockchains enable non-interactive MPC. In *Theory of Cryptography*, 2021.

[33] 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, Heidelberg, August 2022.

[34] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Heidelberg, August 2017.

[35] Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 499–529. Springer, Heidelberg, August 2019.

[36] Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. Constant-round YOSO MPC without setup. *IACR Cryptol. ePrint Arch.*, 2022.

[37] Chen-Da Liu-Zhang, Elisaweta Masserova, João Ribeiro, Pratik Soni, and Sri Aravinda Krishnan Thyagarajan. Efficient distributed randomness generation from minimal assumptions where parties speak sequentially once. *IACR Cryptol. ePrint Arch.*, 2025.

[38] Chen-Da Liu-Zhang, Elisaweta Masserova, João Ribeiro, Pratik Soni, and Sri AravindaKrishnan Thyagarajan. Improved yoso randomness generation with worst-case corruptions. In *International Conference on Financial Cryptography and Data Security*, pages 73–89, 2024.

[39] Jesper Buus Nielsen, João L. Ribeiro, and Maciej Obremski. Public randomness extraction with ephemeral roles and worst-case corruptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 127–147. Springer, Heidelberg, August 2022.

[40] Tal Rabin. New multiparty computational model: From nakamoto to YOSO. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22*, page 1. ACM Press, May / June 2022.

[41] 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, Heidelberg, August 2022.

[42] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

Supplementary Material

A Security Definitions of our Cryptographic Building Blocks

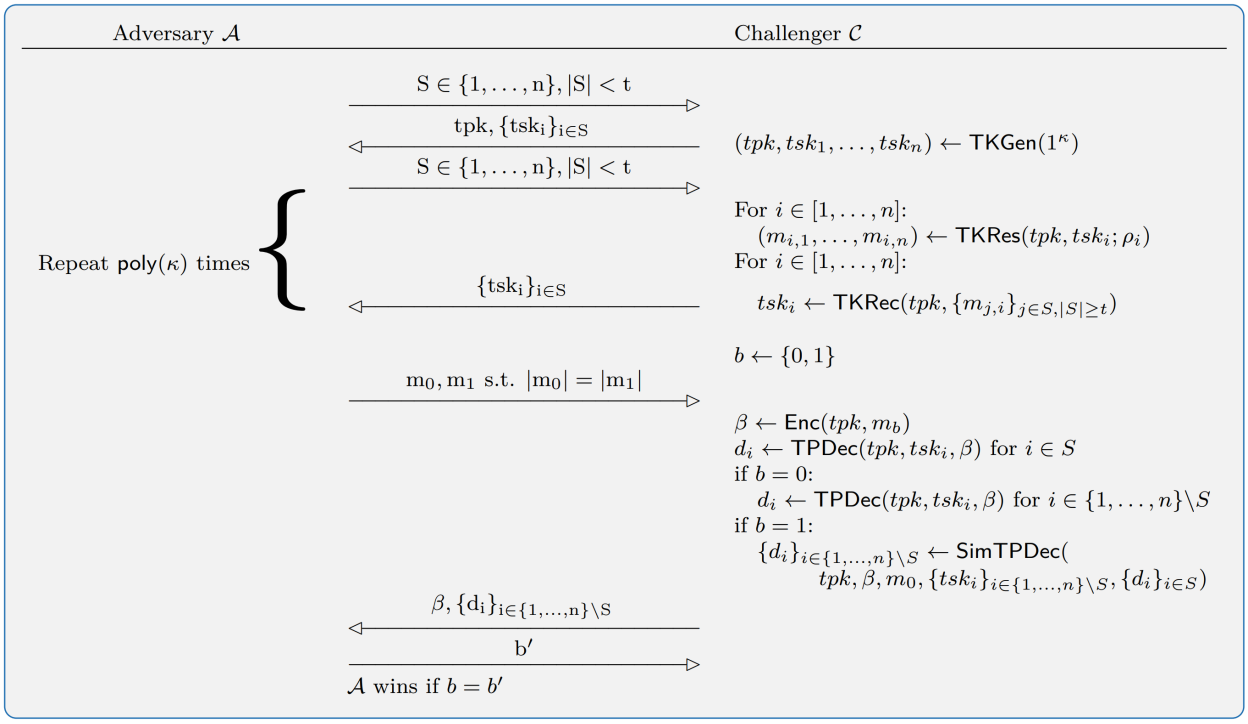
We now formally introduce some security properties of our basic cryptographic building blocks.

A.1 Security of Linearly Homomorphic Key Rerandomizable Threshold Encryption

Below, we formally define the key security property of linearly homomorphic key rerandomizable threshold encryption. Note that it trivially implies chosen plaintext security.

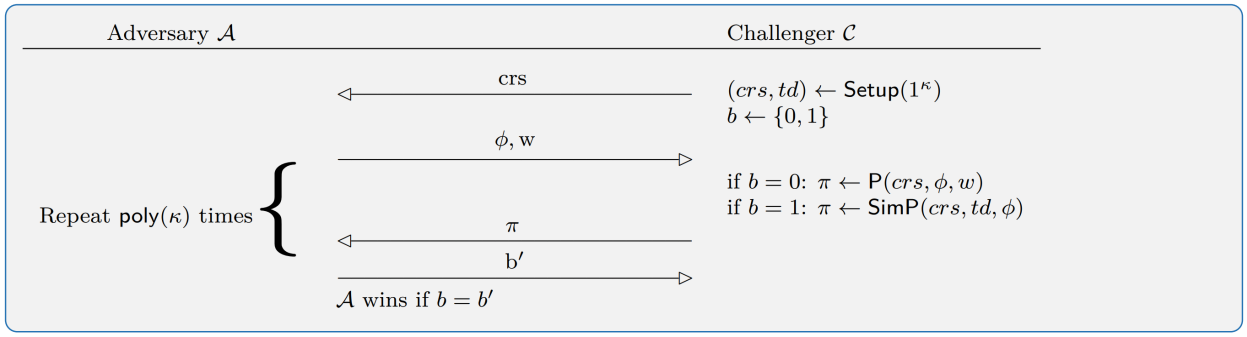
Definition 2 (Partial Decryption Simulatability). Informally, a TE scheme has partial decryption simulatability if for any honestly produced ciphertext, desired message m and fewer than t partial decryptions, the algorithm SimTPDec produces remaining partial decryptions which cause TDec to return m. More formally, let $\kappa \in N$ be the security parameter, and let $\mathsf{TE} = (\mathsf{TKGen}, \mathsf{TEnc}, \mathsf{TPDec}, \mathsf{TDec}, \mathsf{TEval}, \mathsf{TKRes}, \mathsf{TKRec}, \mathsf{SimTPDec})$ be a TE scheme. Consider the game between a probabilistic polynomial-time adversary A and a challenger C described in Figure 2.

Figure 2: Security game for the partial decryption simulatability property of the TE



TE has partial decryption simulatability if for any sufficiently large security parameter κ, for any probabilistic polynomial-time adversary A, there exists a negligible function $negl$ in the security parameter κ such that the probability that A wins the game is less than $\frac{1}{2} + negl(\kappa)$.

Figure 3: Security game for the zero knowledge property of the NIZKAoK



A.2 Security of Non-Interactive Zero-Knowledge Arguments of Knowledge

Definition 3 (Zero Knowledge for NIZKAoK). Informally, a NIZKAoK scheme has zero knowledge if a proof does not leak any more information than the truth of the statement.

More formally, let $\kappa \in \mathbb{N}$ be the security parameter, and let $\mathsf{NIZKAoK} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V}, \mathsf{SimP})$ be a NIZKAoK scheme. Consider the game between a probabilistic polynomial-time adversary A and a challenger C described in Figure 3.

NIZKAoK has zero knowledge if for any sufficiently large security parameter κ, for any probabilistic polynomial-time adversary A, there exists a negligible function *negl* in the security parameter κ such that the probability that A wins the game is less than $\frac{1}{2} + negl(\kappa)$.

Informally, knowledge soundness is the property that guarantees that it is always possible to extract a valid witness from a proof that verifies. Simulation extractability is a stronger version of knowledge soundness, where it is always possible to extract a valid witness from a proof that verifies even if the adversary has access to a simulation oracle. This is a flavor of non-malleability; an adversary should not even be able to modify a simulated proof in order to forge a proof.

Definition 4 (Simulation Extractability for NIZKAoK). Informally, a NIZKAoK scheme has simulation extractability if it is always possible to extract a valid witness from a proof that verifies.

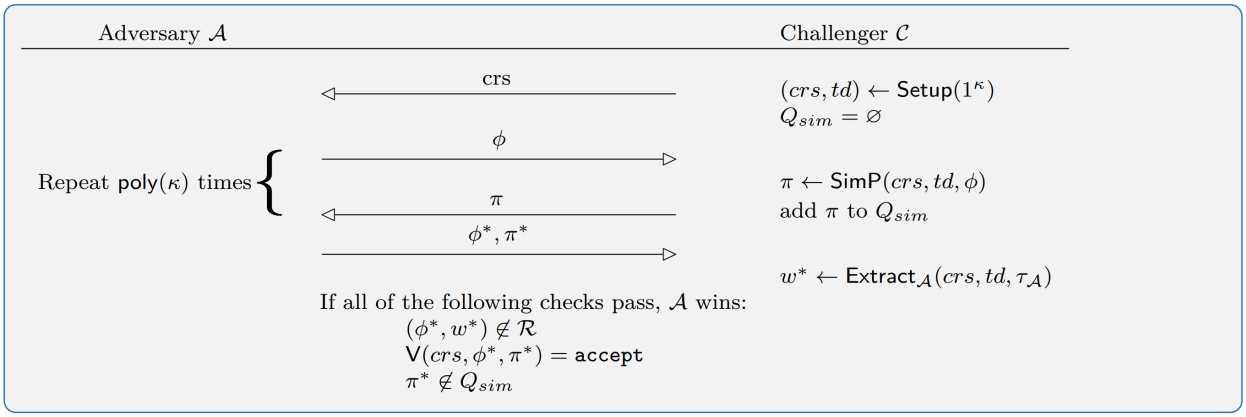
More formally, let $\kappa \in \mathbb{N}$ be the security parameter, and let $\mathsf{NIZKAoK} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V}, \mathsf{SimP})$ be a NIZKAoK scheme. Consider the game between a probabilistic polynomial-time adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ described in Figure 4, where $\tau_{\mathcal{A}}$ denotes the adversary's inputs and outputs, including its randomness:

NIZKAoK has simulation extractability if for any sufficiently large security parameter κ, for any probabilistic polynomial-time adversary A, there exists an extraction algorithm $\mathsf{Extract}_{\mathcal{A}}$ and a negligible function *negl* in the security parameter κ such that the probability that $\mathcal{A}$ wins the game is less than $negl(\kappa)$.

B Security Analysis of Our YOSO MPC

Theorem 2 (Theorem 1, restated)**.** *Assuming a secure broadcast and role-assignment, for any n-party function f, protocol* $\Pi = (\mathsf{YOSO\text{-}Setup}, \Pi_{\mathsf{YOSO\text{-}Offline}}, \Pi_{\mathsf{YOSO-Online}})$*, YOSO-securely implements the ideal functionality* $\mathcal{F}^F_{MPC}$ *for a corruption threshold* $t < \frac{n}{2} \cdot (1 - \epsilon)$*. The offline communication complexity is* $O(n)$ *elements per gate, the online communication complexity is* $O(1)$ *elements per gate.*

Figure 4: Security game for the simulation extractability property of the NIZKAoK



Sketch. We follow a similar approach as the proof from Theorem 2 in the original YOSO paper [29]. As in [29], we provide a sketch of the proof, focusing on the most intricate details involving the YOSO model. For example, intuitive properties such as privacy and correctness follow directly from Turbopack [25]. The nuances in our case are mostly concentrated around the use of threshold decryption, NIZKs, and most importantly the YOSO model, which forces us to be careful when referring to parties, since they are actually computing roles.

In order to describe our simulator, we will describe a series of hybrids, each indistinguishable from the previous one, that eventually lead to a simulator description for the protocol in the theorem.

Hybrid 0: The simulator honestly follows the protocol, except that during the Setup phase the simulator additionally stores the NIZKAoK trapdoor td.

Hybrid 1: The simulator behaves as before, except that it uses the NIZKAoK simulator SimP to simulate the proofs of the honest roles. Note that this hybrid is indistinguishable from the previous one by the zero knowledge property of NIZKAoK system that we use.

Hybrid 2: The simulator behaves as before, except that it now extracts a witness from each proof that was given by a corrupt party. The simulator aborts if it fails to extract a valid witness for a proof that verifies correctly. Note that this hybrid is indistinguishable from the previous one by the simulation extractability property of the NIZKAoK.

Due to this, the simulator in particular always knows the shares of tsk (even for corrupt parties), along with the plaintexts inside the ciphertexts distributed during the offline phase.

Hybrid 3: Let M denote the set of currently corrupt roles. The simulator behaves as before, except that during the $\mathsf{Re\text{-}encrypt*}$ step of computing the output, for each output wire α that belongs to a malicious client, the simulator does the following:

- Let c^α denote the ciphertext that contains the value λ^α encrypted under tpk.
- The simulator computes λ^α, as mentioned above.
- The simulator computes the decryption shares of the corrupt roles as $d_i \leftarrow \mathsf{TPDec}(tpk, tsk_i, c^\alpha)$.
- Compute the decryption shares of the honest roles as $\{d_i\}_{i\in[n]\setminus M} \leftarrow \mathsf{SimTPDec}(tpk, c^\alpha, \lambda^\alpha, \{tsk_i\}_{i\in[n]\setminus M}, \{d_i\}_M)$

Observe that the indistinguishability to the previous hybrid holds by the partial decryption simulatability property of the threshold encryption scheme.

Hybrid 4: Let M denote the set of the currently corrupt roles. The simulator behaves as before, except that for each output wire α that belongs to a malicious client, the simulator does the following:

- Let c^α denote the ciphertext that contains the value λ^α encrypted under tpk.
- The simulator computes the values v^α of the corrupt parties using its knowledge of λ^α, sends these values to the ideal functionality $\mathcal{F}_{\mathsf{MPC}}F$, obtains v^α from it, and computes $\lambda^\alpha = \mu^\alpha - v^\alpha$.
- The simulator computes the decryption shares of the corrupt roles as $d_i \leftarrow \mathsf{TPDec}(tpk, tsk_i, c^\alpha)$.
- Compute the decryption shares of the honest roles as $\{d_i\}_{i\in[n]\setminus M} \leftarrow \mathsf{SimTPDec}(tpk, c^\alpha, \lambda^\alpha, \{tsk_i\}_{i\in[n]\setminus M}, \{d_i\}_M)$

Note that the indistinguishability to the previous hybrid holds by the partial decryption simulatability property of the threshold encryption scheme.

Hybrid 5: Let M denote the set of the currently corrupt roles. The simulator behaves as before, except that for each input wire α that belongs to an honest client, the simulator does the following:

- Let c^α denote the ciphertext that contains the value λ^α encrypted under tpk.
- The simulator generates a fresh random value $\hat{\lambda}^\alpha$.
- The simulator computes the decryption shares of the corrupt roles as $d_i \leftarrow \mathsf{TPDec}(tpk, tsk_i, c^\alpha)$.
- Compute the decryption shares of the honest roles as $\{d_i\}_{i\in[n]\setminus M} \leftarrow \mathsf{SimTPDec}(tpk, c^\alpha, \hat{\lambda}^\alpha, \{tsk_i\}_{i\in[n]\setminus M}, \{d_i\}_M)$

Note that the indistinguishability to the previous hybrid holds as $\hat{\lambda}^\alpha$ and λ^α are both uniformly random, and the threshold encryption scheme has the partial decryption simulatability property.

Hybrid 6: The simulator behaves as before, except that when an honest client needs to publish μ^α, the simulator simply samples a random value μ^α, and publishes it instead. Note that $\mu^\alpha = v^\alpha - \lambda^\alpha$, and from the previous hybrid we know that λ^α is uniformly random. Hence, μ^α is uniformly random and thus the distribution of μ^α remains the same as before.

Note that in our last hybrid, the simulator no longer needs the honest inputs from the input roles. This finishes our proof. □ □

C YOSO Broadcast

For completeness, we give the ideal broadcast functionality $\mathcal{F}_{BC}$ below, as defined by Gentry et al. [30].

Ideal Functionality $\mathcal{F}_{BC}$ for Broadcast

The ideal functionality has the following behavior.

- Initially create a map $y : \mathbb{N} \times \mathsf{Role} \to \mathsf{Msg}_\perp$ with $y(r, \mathsf{R}) = \perp$ for all r, R. Below we use $y(r, \cdot)$ to denote the map $y' : \mathsf{Role} \to \mathsf{Msg}_\perp$ with $y'(\mathsf{R}) = y(r, \mathsf{R})$.
- On input $(\mathsf{Send}, \mathsf{R}, x_\mathsf{R} \in \mathsf{Msg})$ in round r proceed as follows:
  1. Update $y(r, \mathsf{R}) = x_\mathsf{R}$. *Store inputs of the round.*

2. Output $(\mathsf{R}, x_{\mathsf{R}})$ to $\mathcal{S}$. *Leak messages to the simulator in a rushing fashion.*
3. If R is honest then give output Spoke to R.

- On input $(\mathsf{Read}, \mathsf{R}, r')$ in round r where $r' < r$ output $y(r, \cdot)$ to R.