Towards Efficient YOSO MPC Without Setup

Sebastian Kolby[1], Divya Ravi[1], and Sophia Yakoubov[1]*

Aarhus University, Denmark; {sk, divya, sophia.yakoubov}@cs.au.dk

Abstract. YOSO MPC (Gentry *et al.*, Crypto 2021) is a new MPC framework where each participant can speak at most once. This models an adaptive adversary's ability to watch the network and corrupt or destroy parties it deems significant based on their communication. By using private channels to anonymous receivers (e.g. by encrypting to a public key whose owner is unknown), the communication complexity of YOSO MPC can scale sublinearly with the total number N of available parties, even when the adversary's corruption threshold is linear in N (e.g. just under $N/2$). It was previously an open problem whether YOSO MPC can achieve guaranteed output delivery in a constant number of rounds without relying on trusted setup. In this work, we show that this can indeed be accomplished. Using linearly homomorphic encryption and secret sharing, we construct YOSO-LHE, which is the first realistically efficient YOSO MPC protocol that achieves guaranteed output delivery without trusted setup. YOSO-LHE is not itself constant-round; it takes $O(d)$ rounds of communication, where d is the depth of the circuit being computed. However, YOSO-LHE can be used to bootstrap any constant-round YOSO protocol that requires setup, by generating that setup within YOSO-LHE. As long as the complexity of the setup is independent of the circuit to be evaluated, the bootstrapped protocol will be constant-round.

*Funded in part by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO) and No 803096 (SPEC).

Table of Contents

1 Introduction

As our digital world becomes more reliably connected, we grow to depend more and more on outsourcing our data storage and processing to the cloud. There are clear benefits to doing this, such as minimizing the risk of data loss (e.g. when we spill coffee on our laptops), and using resources more efficiently. However, there are also serious drawbacks, such as having to trust a cloud provider to maintain both the availability and privacy of our data in an era of frequent data breaches. Secure multi-party computation (MPC) [CCD88,GMW87,Yao86] allows a cloud comprised of many distinct machines to not only store, but also process our data securely. MPC guarantees that the data remains private even from an attacker controlling fewer that some threshold t of the machines.

Outsourcing the processing of our data can be very useful: for instance, it lets us search our securely stored emails without having to download the entire contents of our inbox. Perhaps even more importantly, MPC can be used to compute joint functions on multiple entities' private data without revealing anything but the function output. This enables crucial computations where multiple entities have privacy concerns, like research using health data across different hospital databases [VCAZ$^+$18,sod19], and discovering the true extent of the gender wage gap across many different institutions [LJA$^+$18].

One long-standing challenge in MPC is balancing security and efficiency. Intuitively, security increases with the number of machines we employ in our

MPC: the more machines are used, the more of them an attacker would need to subvert in order to learn our data. In order to make our computation secure against a powerful attacker, we would need a very large number of machines — perhaps millions, e.g. in a distributed blockchain setting. However, running a secure computation among millions of machines would be horribly inefficient; transferring our data to those machines would be a prohibitive burden on our devices, and the pairwise communication required by most MPC protocols would be too much for the machines and their network.

An alternative path is to hide which machines we are outsourcing our data to. Then, a small subset of machines could perform the computation, bypassing the problem of prohibitive pairwise communication. By keeping the subset anonymous, we ensure that an attacker won't know which machines to target.

This approach is very tricky, since computing on data requires the machines to communicate. However, as soon as a machine sends a message, it ceases to be anonymous; an attacker who is watching the network will learn that the message-sender — and message-receiver — likely play crucial roles in the computation, and will be able to target them. The recent work of Gentry *et al.* [GHK$^+$21] introduces secure computation in the you only speak once (YOSO) model. In this model, once a machine sends a message, that machine becomes irrelevant to the computation, so an attacker who then targets that machine gains nothing. To stop an attacker from targeting the message recipients, YOSO protocols hide their identities using what we call receiver-anonymous communication channels.

1.1 Related Work

Related work can be broken up into work focusing on receiver-anonymous communication channels (Section 1.1) and on MPC protocols using those channels (Section 1.1).

Receiver-Anonymous Communication Channels When data is outsourced to a small set of machines, those machines immediately assume critical roles; so, it is important to hide which machines are picked for this by choosing them randomly and by keeping them anonymous. Of course, outsourcing data to a set of machines requires private communication to those machines. In order to communicate privately to machines which must remain anonymous, we need *receiver-anonymous communication channels* (RACCs) to those machines. We model RACCs as a publicly known encryption keys such that (a) the adversary does not know who owns the corresponding decryption key as long as that owner is not corrupt, and (b) fewer than some threshold t (which we take to be half) of the decryption key owners are corrupt.

Receiver-anonymous communication channels first appeared in the work of Benhamouda *et al.* [BGG$^+$20], which builds such channels with the guarantee that only half of the decryption key owners are corrupt as long as only around a quarter of the overall population is corrupt. Another RACC construction was shown by Gentry *et al.* [GHM$^+$21], with the stronger guarantee that only half of the decryption key owners are corrupt as long as only slightly less than half of the overall population is corrupt. The downside of this second construction is that it is more computationally intensive.

In this work, we do not focus on the problem of setting up RACCs, and instead study how to build YOSO MPC on top of them.

YOSO MPC Recently, the first YOSO MPC protocols have appeared in the literature [GHK$^+$21,CGG$^+$21]. They all have a common structure: committees of size n — where the size n is chosen to guarantee that n RACCs will have an honest majority with overwhelming probability — carry out the computation sequentially. The lth (set of) committee(s) performs the lth layer of multiplication, and uses RACCs to pass the computation to their successors.

We summarize the constructions below, and compare them in Figure 1. It should be noted that even constructions which do not explicitly make any computational assumptions rely on RACCs, which do require such assumptions.

Fluid MPC (Choudhuri *et al.* [CGG$^+$21]) This construction relies on specially tailored sum-checks on top of the BGW construction [BGW88]. While concretely efficient, Fluid MPC does not guarantee output delivery; that is, a single corrupt participating machine can cause the entire protocol to abort, allowing the adversary to carry out a denial of service attack indefinitely.

YOSO-CDN (Gentry *et al.* [GHK$^+$21]) This construction is based closely on the CDN protocol [CDN01]. Unlike Fluid MPC, CDN does guarantee output delivery, and is thus robust against denial of service attacks. However, this protocol requires computational assumptions and *setup*; that is, some correlated secrets are distributed to an initial subset of parties by e.g. a trusted authority.

YOSO-IT (Gentry *et al.* [GHK$^+$21]) This construction relies on new information-theoretic techniques such as future broadcast, distributed commitments, and augmented verifiable secret sharing. Like CDN, YOSO-IT guarantees output delivery. However, while YOSO-IT does not require computational assumptions or setup, the number of committees required for each layer of multiplication is polynomial rather than constant in the committee size, which can be prohibitively inefficient.

YOSO-CDN. Since our protocol is closely related to the YOSO-CDN protocol of Gentry *et al.* [GHK$^+$21], we recap CDN — and YOSO-CDN— here.

CDN [CDN01] relies on linearly-homomorphic threshold encryption (LHTE), where a fixed public encryption key pk is known, and the corresponding secret decryption key sk is secret shared among the n participants. A party can supply an input by encrypting it under pk, and publishing the resulting ciphertext. The participants then perform the computation by leveraging the homomorphism of the encryption scheme for linear operations, and by using Beaver triples for multiplications.

YOSO MPC scheme	Security Guarantee	Number of Rounds	Number of Speakers	Setup	Computational Building Blocks
Fluid MPC [CGG+21]	Security with Abort	$O(d)$	$O(dn)$	**none**	**none**
YOSO-CDN [GHK+21]	**Guaranteed Output Delivery**	$d+3$	$m+(d+1)n+2dh$	CRS, distribution of shares to first committee	NIZK, LHTE
YOSO-IT [GHK+21]	**Guaranteed Output Delivery**	$O(d)$	$\mathsf{poly}(m,d,n)$	**none**	**none**
YOSO-LHE (this work)	**Guaranteed Output Delivery**	$d+4$	$m+(d+2)n+(2d+1)h$	**none**	Multi-String NIZK, LHE
Bootstrapping (this work)	**Guaranteed Output Delivery**	$\mathbf{O(1)}$	$\mathbf{O(n)}$	**none**	Multi-String NIZK, LHE, TFHE

Fig. 1: YOSO MPC Constructions. d is the multiplicative depth of the circuit being computed; m is the number of inputs; n is the committee size chosen to guarantee that n RACCs will have an honest majority with overwhelming probability; h is the committee size chosen to guarantee that h RACCs will have at least one honest receiver with overwhelming probability. Properties that are optimal are in **bold**.

Assuming the availability of a Beaver triple $\overline{a} = \mathsf{Enc}(a)$, $\overline{b} = \mathsf{Enc}(b)$ and $\overline{c} = \mathsf{Enc}(ab)$, the parties multiply ciphertexts $\overline{x}$ and $\overline{y}$ (encrypting x and y, respectively) by (1) using the linear homomorphism to compute $\overline{\epsilon} = \overline{a-x}$ and $\overline{\delta} = \overline{b-y}$, (2) jointly decrypting ϵ and δ, and (3) using linear homomorphism to compute $\overline{xy} = \overline{c - \epsilon b - \delta a + \epsilon\delta}$.

The Beaver triples themselves can be generated on-the-fly in two rounds. In the first round, each participant i of that round chooses a random additive share a_i of a, and publishes $\overline{a_i} = \mathsf{Enc}(a_i)$ together with a zero knowledge proof that it is well-formed. Everyone can then use the linear homomorphism to compute $\overline{a} = \overline{\sum a_i}$, using only the contributions $\overline{a_i}$ which are accompanied by a verifying proof. In the second round, each participant i of that round similarly contributes an encrypted additive share $\overline{b_i}$ of b, together with $\overline{b_i a}$ (which she computes as a linear operation on $\overline{a}$ using her knowledge of b_i), and a zero knowledge proof that $\overline{b_i}$ and $\overline{b_i a}$ were produced consistently. Everyone can then compute $\overline{b} = \overline{\sum b_i}$ and $\overline{c} = \overline{\sum b_i a}$ using the contributions from those parties i whose zero knowledge proofs verify.

To YOSO-ify this construction, Gentry *et al.* needed to make only a few minor changes to ensure that every round of communication can be carried out by a new committee. First, they observe that the two rounds of communication that generate a Beaver triple (a) do not depend on the shared secret decryption key, and so (b) can be carried out by committees with a dishonest majority and no RACCs. They thus instruct two smaller committees of size h (where h is chosen to guarantee that a set of h random parties will contain at least one honest party with overwhelming probability) to carry out Beaver triple generation.

All that remains is to ensure that *every* committee that must decrypt a value (whether those values are the ϵ and δ needed for a multiplication, or the computation output itself) holds shares of the secret decryption key. In order to do this, we need an additional property from the threshold linearly homomorphic encryption scheme: it must allow a committee that holds a sharing of the secret decryption key to re-share that key to the next committee in a single round of communication. (They can do this in the same breath in which they broadcast their contributions to the decryption of the values they are opening.) Gentry *et al.* use an encryption scheme that has this property. An unfortunate downside of this is that each secret key share has size $O(n)$, resulting in $O(n^2)$ communication per committee member as part of the key resharing (even disregarding the size of the accompanying zero knowledge proof). An even more important remaining issue is how the public encryption key, together with the initial sharing of the decryption key, is generated. YOSO-CDN relies on a trusted setup for this.

1.2 Our Contributions

In this paper, we describe two YOSO MPC protocols which have guaranteed output delivery and do not require setup. We call the first of these *YOSO-LHE*; it is based on YOSO-CDN [GHK+21], but avoids the use of a secret shared decryption key, which is the setup that YOSO-CDN relies on. YOSO-LHE takes $O(d)$ rounds, where d is the multiplicative depth of the circuit being computed.

A question that was previously open is whether it is possible to have a YOSO MPC protocol without setup where the number of rounds of communication is independent of the circuit being computed. We answer this question in the affirmative, by using YOSO-LHE to execute the setup for a constant-round protocol that uses threshold fully homomorphic encryption (TFHE). We call this the *bootstrapping* protocol.

1.3 YOSO-LHE: Technical Overview

YOSO-LHE is based closely on the YOSO-CDN protocol, described above. However, we make a crucial pivot: instead of using a threshold linearly homomorphic encryption scheme (LHTE) with a global public key, we layer linearly homomorphic threshold secret sharing (e.g. Shamir secret sharing) and linearly homomorphic encryption (LHE) with participants' *individual* (RACC) public keys, which eliminates the need for a trusted setup.

In more detail, in order to provide an input x to the computation, instead of encrypting x to a global public key as in YOSO-CDN, a party must first secret share x as $(x_1, \ldots, x_n)$, and then encrypt each share to a member of a specific committee. That committee now holds a sharing of x, and can either jointly decrypt x at the appropriate time (by decrypting and publishing the shares), or compute on x and other values it may hold. Linear computations accomplished by leveraging the linear homomorphisms of both the secret sharing and encryption schemes; a multiplication requires the use of a Beaver triple, just like in YOSO-CDN. However, the Beaver triple must now be generated for this specific committee, as shares encrypted under its public encryption keys.

Unlike in CDN, if a value is input to one committee but must later be used by a different committee, the first committee must re-share the value to the new committee. This can be done simply by having each party re-share its share to the new committee, and prove in zero knowledge that it did so correctly.

Eliminating Setup Recall that the YOSO-CDN protocol requires two types of setup (ignoring the RACC channels, which are an inherent requirement for all YOSO protocols):

1. The generation of the LHTE global public key and the distribution of shares of the associated decryption secret key to the first committee.
2. The common reference string (CRS) for the NIZKs.

Using (encrypted) secret sharing instead of LHTE eliminates the first form of setup. We can eliminate the need for a CRS by using a *multi-string NIZK* [GO07] instead of a regular NIZK; such a NIZK uses multiple common reference strings, and provides the desired guarantees as long as at least half of these strings were produced honestly. By relying on a committee with an honest majority to produce these strings, we avoid the need to assume a CRS provided by a trusted setup.

Choosing Compatible Linearly Homomorphic Encryption and Secret Sharing The natural choice of linearly homomorphic secret sharing is Shamir secret sharing [Sha79]. We have several constraints for picking our homomorphic encryption scheme: (a) it must offer a linear homomorphism over the *same* finite field for independently generated key pairs (in order to support operations over shares from a single secret sharing), and (b) it must not require a common reference string, which we just took care to eliminate. In the famous linearly homomorphic encryption scheme due to Paillier [Pai99], the message space — integers modulo n, where n is a product of two primes — is inherently different from key to key, since the factorization of n can act as the secret key. The variant of the Paillier cryptosystem due to Bresson *et al.* [BCP03] — which has an ElGamal-like structure — allows generating multiple key pairs with the same message space, but at the expense of requiring a CRS; here, n can be considered a public paramterer, but it must be generated in a trusted way, so that the factorization of n is unknown to anyone.

We instead use the linearly homomorphic cryptosystem of Castagnos and Laguillaumie [CL15], which also has an ElGamal-like structure. The plaintext m is encoded in an exponent (as f^m) during encryption, so that the natural multiplicative homomorphism of ElGamal becomes an additive one. In order to enable efficient decryption, Castagnos and Laguillaumie use class groups, and encode m using a generator f of a subgroup where the discrete logarithm problem is efficiently solvable (much like in the Paillier-based cryptosystems, but without a second trapdoor in the form of the factorization of n). The message space will be integers modulo a prime p, where p can be a fixed parameter across multiple independently generated key pairs (under the constraint that p is big enough).[1]

Assumptions YOSO-LHE inherits its assumptions from the linearly homomorphic encryption scheme and multi-string NIZK scheme it uses. The encryption scheme of Castagnos and Laguillaumie [CL15] is based on the DDH assumption in the class group setting; the Multi-string NIZKs due to Groth and Ostrovsky [GO07] are based on enhanced trapdoor permutations.

Communication Complexity, Round Complexity and Future Horizon YOSO-LHE uses two types of committees: committees of size n (where the size n is chosen to be large enough to guarantee an honest majority within the committee), and committees of size h (where the h can be much smaller than n, since only one honest party, rather than an honest majority, is needed). YOSO-LHE uses one committee of size n for each layer of multiplication, as well as two additional committees of size n: one to create the multi-string NIZK common reference strings, and one to decrypt the output. Each multiplication requires two committees of size h to generate a Beaver triple. One additional committee of size h must generate a sharing of zero to mask the output. Including m parties who speak to provide inputs to the computation, this makes the total number of parties who speak throughout the protocol equal to $m + (d+2)n + (2d+1)h$.

It might look like this necessitates $3d+4$ rounds of communication, but many of these committees can speak at the same time. The committee creating the common reference strings must speak first, and the two committees generating Beaver triples for the first multiplication must both speak before the first multiplication can happen, but the parties providing input can speak at the same time as one of those Beaver triple committees. Committees generating Beaver triples for future multiplications can always speak in parallel with previous committees; in fact, if desired, all of the Beaver triple committees could speak at once, or alternatively, a single committee could generate all of the Beaver triples (and the sharing of zero to mask the output). Of course, the committee who decrypts the output must speak last, which leaves us with $d+4$ rounds of communication.

One reason not to have a single pair of committees of size h generate all of the Beaver triples is what Gentry *et al.* call the *future horizon*, which describes how long before a role needs to act must her receiver anonymous communication channel be available, or, in other words, how far in advance must machines be assigned to their roles, or how many rounds can separate a speaker i from the last speaker j to whom speaker i must send a message (by encrypting to the RACC key pk_j). It is desirable to minimize the future horizon, because when running in e.g. a blockchain environment, the pool of participants can be very dynamic, and because machines are always coming and going it can be impractical to select machines for roles too far in advance (since they might disappear before the time comes). A small future horizon enables on-the-fly assignment of machines

[1] p is not a CRS, since security does not depend on the honest choice of p.

to roles while the protocol runs. If a single pair of committees generates all of the Beaver triples, the future horizon of YOSO-LHE will be determined by the round distance from the first of these two Beaver committees to the last committee that receives an output of a multiplication (which will be the output committee). This distance will be $d + 2$. If instead we have a designated pair of committees generate Beaver triples for each layer of the multiplication (as we described above), the future horizon is 3. (We assume that the output wires of a given layer of multiplication gates in the circuit being computed serve as input only to the next layer of multiplication gates; otherwise, the future horizon is determined by the longest wire in the circuit. Note that any circuit can be converted to a circuit with the property we assume by adding multiplication-by-one gates.)

Comparison to YOSO-CDN There are advantages and disadvantages to the changes we make to YOSO-CDN to obtain YOSO-LHE. Of course, a crucial advantage of YOSO-LHE— and the motivation for our changes — is that YOSO-LHE does not require trusted setup.

On the other hand, a disadvantage of YOSO-LHE is that a given value is held by one committee; for it to be used by several committees, it must be shared to several, or re-shared from committee to committee. In YOSO-CDN, any value encrypted to the global public key is accessible by *any* committee that holds the shared secret decryption key; no additional work to make the value accessible to a given committee is needed. In particular, this means that in YOSO-LHE, Beaver triple preprocessing must be done with a committee in mind. In YOSO-CDN, all Beaver triples are useable by any committee.

1.4 Achieving Setup-Free Constant-Round YOSO MPC

It was previously an open problem to describe a setup-free constant-round YOSO MPC. Gentry *et al.* [GHK$^+$21] point out a simple constant-round YOSO MPC protocol: if a secret key for a threshold *fully* homomorphic encryption (FHE) scheme is shared to a committee, that committee can perform the entire computation as long as joint decryption only requires a single round of communication. However, this protocol requires setup, in the form of the distribution of the secret key shares.

We observe that it is possible to combine any setup-free YOSO MPC with any constant-round YOSO MPC (which might rely on setup) to obtain a setup-free constant-round YOSO MPC. This can be done simply by performing the setup for the constant-round YOSO MPC within the setup-free YOSO MPC, and then using the constant-round YOSO MPC for the actual computation. If the setup performed by the setup-free YOSO MPC is independent of the circuit we wish to compute, the number of rounds required by this bootstrapped protocol will be independent of the size of the circuit as well. We can use our setup-free YOSO MPC — YOSO-LHE— to generate a threshold fully homomorphic encryption key, and share the corresponding decryption key to a committee.

(It should be noted that the YOSO-IT construction of Gentry *et al.* could also be used as the setup-free YOSO MPC here; however, that construction is much less practically efficient than ours, due to the large number of committees they require even for a single multiplication.)

1.5 Remark about RACCs

Notice that in YOSO-LHE, we use receiver-anonymous communication channels (RACCs) in a non-black box way: we rely on the homomorphic properties of the encryption schemes realizing the RACCs, and sometimes we require zero knowledge proofs about the encrypted messages. In this sense, we stray from the ideal RACCs described by Gentry *et al.* [GHK+21]. However, YOSO-CDN [GHK+21] already strays similarly; while they do not require the encryption scheme realizing the RACCs to be homomorphic, they do need zero knowledge proofs about encrypted messages. The properties required by YOSO-CDN and YOSO-LHE are easily provided by existing realizations of RACCs [BGG+20,GHM+21].

2 YOSO Secure Multiparty Computation (MPC) Definitions

In this section we recap what it means for an MPC protocol to be YOSO secure. The YOSO model [GHK+21] makes a crucial separation between physical machines and the roles which they play in the protocol. By mapping machines to roles in a random and unpredictable way, we can ensure that the adversary will not know which machines will be important, and will not be able to preemptively corrupt or destroy those machines. In this paper, we describe our YOSO MPC protocols in terms of roles. We ignore how roles are assigned to machines; we assume the availability of a role assignment functionality which publishes encryption keys which can be used to communicate with future roles. Mechanisms which realize such a role assignment functionality were described by Benhamouda *et al.* [BGG+20] and Gentry *et al.* [GHM+21]. (Our protocols assume that the encryption scheme used has some special properties; the role assignment mechanisms cited above support this.)

The YOSO model uses the UC framework [Can01], with roles instead of physical machines as the participants. Every participant is 'YOSO-ified', meaning that as soon as she speaks for the first time, she is killed. A protocol Π YOSO-realizes a functionality $\mathcal{F}$ if the YOSO-ification of Π UC-realizes $\mathcal{F}$ (Section 2.1).

2.1 UC MPC

Consider a protocol $\Pi = (\mathsf{R}_1, \ldots, \mathsf{R}_u)$ described as a tuple of roles R_i, each of which is a probabilistic polynomial-time (PPT) machine. Some of those roles are *input* roles, who, when they speak, provide an input. Other roles are there to assist in computing a function f on the provided inputs.

In a real-world execution of protocol Π on input $x = (x_1, \ldots, x_m)$ with environment $\mathcal{E}$ and adversary $\mathcal{A}$, the PPT environment $\mathcal{E}$ is given the input x, and in turn provides the input to protocol's input roles. The environment also communicates with the PPT adversary $\mathcal{A}$. We consider a *synchronous* model, where the protocol is executed in rounds; in each round, some roles speak (over a broadcast channel). During the execution of the protocol, the corrupt roles receive arbitrary instructions from $\mathcal{A}$, while the honest parties faithfully follow the instructions of the protocol. We consider the adversary $\mathcal{A}$ to be rushing, i.e., during every round the adversary can see the messages the honest roles sent before producing messages from corrupt roles. At the end of the protocol execution, the environment $\mathcal{E}$ produces a binary output. Let $REAL_{\Pi,\mathcal{A},\mathcal{E}}(x)$ denote the random variable (over the random coins used by all roles) representing $\mathcal{E}$'s output in the real world.

Now, consider an ideal-world execution on the same input x and with the same environment $\mathcal{E}$, but with an ideal-world adversary $\mathcal{S}$. In the ideal-world execution, instead of running the protocol Π, the roles turn to a trusted party to compute f. This trusted party receives the inputs $x_1, \ldots, x_m$ from the input roles, and broadcasts $f(x_1, \ldots, x_m)$. We call this trusted party the *ideal functionality* $\mathcal{F}_f$. Let $IDEAL_{\mathcal{F}_f,\mathcal{S},\mathcal{E}}(x)$ denote the random variable (over the random coins used by all roles) representing $\mathcal{E}$'s output in the ideal world.

Definition 1 (UC Security [Can01]). *Let $f : (\{0,1\}^*)^m \rightarrow \{0,1\}^*$ be an m-input function. A protocol $\Pi = (\mathsf{R}_1, \ldots, \mathsf{R}_u)$ UC-securely computes f (with guaranteed output delivery) if for every PPT real-world adversary $\mathcal{A}$ there exists a PPT ideal-world adversary (or* simulator*) $\mathcal{S}$ such that, for any PPT environment $\mathcal{E}$, it holds that $REAL_{\Pi,\mathcal{A},\mathcal{E}}(x)$ and $IDEAL_{\mathcal{F}_f,\mathcal{S},\mathcal{E}}(x)$ are indistinguishable for any set of inputs $x = (x_1, \ldots, x_m)$.*

2.2 The YOSO Adversary's Corruption Power

Gentry *et al.* show that, given a role assignment mechanism that randomly maps roles to machines, an adversary with the ability to selectively corrupt machines corresponds to an adversary who *randomly* corrupts roles. This lets us assume that an adversary who can corrupt slightly fewer than half of the available machines can corrupt less than half of the roles in a *committee* of roles as long as the committees are chosen to be large enough. We let n be the committee size that ensures an honest majority of roles. We let h be the (smaller) committee size that ensures at least one honest role on the committee.

Gentry *et al.* also point out that for random corruptions, there is very little difference between adaptive corruptions and static corruptions. In the case of random corruptions, the adversary must leave the choice of which party to corrupt to a special *corruption controller*; the adversary cannot tell whether the corruption controller makes this random choice on the fly, or whether the choice was made before the start of the protocol. We follow the path laid out by Gentry *et al.*, and phrase our proof in terms of static security, noting that it can be extended to the adaptive case using standard techniques.

Roles which are expected to provide input are a special case, since it makes no sense to request input from random machines; rather, there are likely predetermined participants who are expected to provide meaningful inputs. We prove our protocols secure without making any assumptions about the adversary's ability to corrupt input roles. In particular, we do not require an honest majority of input parties.

To summarize, we prove security against an adversary who can statically corrupt (a) arbitrarily many input roles, (b) fewer than half of the roles in each committee of size n, and (c) all but one of the roles in each committee of size h.

3 YOSO-LHE

In this section, we describe our YOSO MPC protocol based on linearly homomorphic encryption (LHE). We assume that role assignment has been executed; that is, we assume the availability of an encryption key pk_{R} (for a linearly homomorphic encryption scheme) for every role R, such that the appropriate machine knows the corresponding decryption key sk_{R}.

We start with an informal overview. As a first step, we have a committee of roles each locally compute a CRS for the multi-string NIZK scheme. If we pick our committee size n in such a way that at least half the committee roles are honest, then a majority of the CRS's will be honestly generated, which is enough to securely use the multi-string NIZK scheme. This step can be executed once and for all, and does not need to be re-executed for each computation.

Next, to submit an input x to a committee C of size n, an input owner first Shamir secret shares that input with threshold $n/2$ as $(x_1, \ldots, x_n)$. She then encrypts each share x_i to one of the members of C as $\overline{x_i}$.

To perform linear operations, the members of a committee use the linear homomorphism of both Shamir secret sharing and of the encryption scheme. Performing multiplications is more involved. Let M denote the committee which holds shares of the values to be multiplied, and let O denote the committee to which we would like to give shares of the products. The multiplication requires two additional committees A and B — each of which only needs to have one honest party, as opposed to an honest majority — to generate a Beaver triple. First, each member A_j of committee A chooses a random value a_j, shares and encrypts it *to both committees M and O*, and broadcasts the two sets of encrypted shares (together with zero knowledge proofs that they executed these steps correctly). The value a is defined as the sum of the a_j's accompanied by verifying proofs. Encryptions $\overline{a_i}$ of shares of a held by members of M and encryptions $\overline{a_i}'$ of shares of a held by members of O are now publicly computable using the linear homomorphisms of the secret sharing and encryption schemes.

Each member B_j of committee B then chooses a random value b_j, and proceeds similarly to the members of committee A, except each party B_j also computes a ciphertext $\overline{b_j a}$ (by multiplying $\overline{a}$ by b_j, which it knows), and broadcasts that ciphertext together with $\overline{b_j}$. As with a, b is defined as the sum of the b_j's accompanied by verifying proofs; $c = ab$ is similarly defined as the sum of $b_j a$'s accompanied by verifying proofs. Encryptions of shares of b and c are again publicly computable using the linear homomorphisms of the schemes.

Next, to multiply two encrypted and shared values x and y using the generated Beaver triple (a, b, c), committee M locally computes shares of $\epsilon = a - x$ and $\delta = b - y$ and broadcasts these shares, allowing public reconstruction of ϵ and δ. Now that committee M has spoken, committee O picks up the torch. They use their own encrypted shares of a, b and c, as well as the reconstructed ϵ and δ, to compute shares of $xy = c - \epsilon b - \delta a + \epsilon\delta$. (Note that we use this version of the Beaver triple arithmetic — avoiding using shares of x and y — since shares of x and y were held by committee M, and may by default not be available to members of committee O.) Finally, before decrypting the output, we mask the output with a sharing of zero.

3.1 Formal Description of YOSO-LHE

We describe the formal protocol below. Our construction uses the following tools:

– A Shamir secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$, described in Appendix A.1.
– Linearly homomorphic public key encryption $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{KeyMatches})$, described in Appendix A.2.
– Multi-string NIZK proofs $(\mathsf{mNIZK.Gen}, \mathsf{P}, \mathsf{V})$, described in Appendix A.3.

We use multi-string NIZK proofs for the following relations:

$$\mathcal{R}_{\mathsf{Share}} = \left\{ \begin{array}{l} \phi = \big(j, \{pk_i\}_{i\in[n]}, \{\overline{x_i}\}_{i\in[n]}\big) \\ w = \big(x, \rho, \{\rho'_i\}_{i\in[n]}\big) \end{array} \middle| \begin{array}{l} (x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x; \rho) \\ \wedge\ \overline{x_i} \leftarrow \mathsf{Enc}(pk_i, x_i; \rho'_i) \text{ for } i \in [n] \end{array} \right\},$$

$$\mathcal{R}_{\mathsf{Dec}} = \left\{ \begin{array}{l} \phi = \big(pk, x, \overline{x}\big) \\ w = \big(sk\big) \end{array} \middle| \begin{array}{l} \mathtt{accept} \leftarrow \mathsf{KeyMatches}(pk, sk) \\ \wedge\ x \leftarrow \mathsf{Dec}(sk, \overline{x}) \end{array} \right\},$$

$$\mathcal{R}_{\mathsf{Beaver,A}} = \left\{ \begin{array}{l} \phi = \begin{pmatrix} l, j, \{pk_{M,i}\}_{i\in[n]}, \\ \{pk_{O,i}\}_{i\in[n]}, \{\overline{a_i}\}_{i\in[n]}, \\ \{\overline{a_i}'\}_{i\in[n]} \end{pmatrix} \\ w = \begin{pmatrix} a, \rho, \rho', \{\rho_{M,i}\}_{i\in[n]}, \\ \{\rho_{O,i}\}_{i\in[n]} \end{pmatrix} \end{array} \middle| \begin{array}{l} (a_1, \ldots, a_n) \leftarrow \mathsf{Share}(a; \rho) \\ \wedge\ (a'_1, \ldots, a'_n) \leftarrow \mathsf{Share}(a; \rho') \\ \wedge\ \overline{a_i} \leftarrow \mathsf{Enc}(pk_{M,i}, a_i; \rho_{M,i}) \text{ for } i \in [n] \\ \wedge\ \overline{a_i}' \leftarrow \mathsf{Enc}(pk_{O,i}, a'_i; \rho_{O,i}) \text{ for } i \in [n] \end{array} \right\},$$

$$\mathcal{R}_{\mathsf{Beaver,B}} = \left\{ \begin{array}{l} \phi = \begin{pmatrix} l, j, \{pk_{M,i}\}_{i\in[n]}, \\ \{pk_{O,i}\}_{i\in[n]}, \{\overline{b_i}\}_{i\in[n]}, \\ \{\overline{b_i}'\}_{i\in[n]}, \{\overline{c_i}'\}_{i\in[n]}, \\ \{\overline{a_i}'\}_{i\in[n]} \end{pmatrix} \\ w = \begin{pmatrix} b, \rho, \rho', \{\rho_{M,i}\}_{i\in[n]}, \\ \{\rho_{O,i}\}_{i\in[n]} \end{pmatrix} \end{array} \middle| \begin{array}{l} (b_1, \ldots, b_n) \leftarrow \mathsf{Share}(b; \rho) \\ \wedge\ (b'_1, \ldots, b'_n) \leftarrow \mathsf{Share}(b; \rho') \\ \wedge\ \overline{b_i} \leftarrow \mathsf{Enc}(pk_{M,i}, b_i; \rho_{M,i}) \text{ for } i \in [n] \\ \wedge\ \overline{b_i}' \leftarrow \mathsf{Enc}(pk_{O,i}, b'_i; \rho_{O,i}) \text{ for } i \in [n] \\ \wedge\ \overline{c_i}' := \mathsf{Eval}(pk_{O,i}, \overline{a_i}', b) \text{ for } i \in [n] \end{array} \right\},$$

$$\mathcal{R}_{\mathsf{zero}} = \left\{ \begin{array}{l} \phi = \big(j, \{pk_i\}_{i\in[n]}, \{\overline{x_i}\}_{i\in[n]}\big) \\ w = \big(\rho, \{\rho'_i\}_{i\in[n]}\big) \end{array} \middle| \begin{array}{l} (x_1, \ldots, x_n) \leftarrow \mathsf{Share}(0; \rho) \\ \wedge\ \overline{x_i} \leftarrow \mathsf{Enc}(pk_i, x_i; \rho'_i) \text{ for } i \in [n] \end{array} \right\},$$

The statement for each of the relations contains a description of the role who will be proving that statement. For $\mathcal{R}_{\mathsf{Share}}$, this description is an index j of an input role. For relation $\mathcal{R}_{\mathsf{Dec}}$, this description is the role's public encryption key (as appointed to it by the role assignment mechanism). For the Beaver relations, this description is the number l of the multiplication, and the index of the j of the committee member. Having statements that necessarily differ from role to role prevents roles from copying and replaying others' proofs. In particular, this prevents input roles from copying others' inputs (possibly without knowing the input they are copying). To avoid such replay attacks accross different executions of the protocol, it is also important to include a session identifier (sid) in all of the statements. We leave this implicit to keep notation simple.

Below, we describe our notation for the relevant roles. For each role R for which a receiver anonymous communication channel (RACC) is available, we let pk_{R} denote that role's public encryption key, and sk_{R} denote that role's private decryption key.

- $T_1, \dots, T_n$ denotes the roles of the preparation committee (of size n) responsible for preparing the common reference strings to be used in the multi-string NIZK proof system.
- $C_{l,1}, \dots, C_{l,n}$ denotes the roles of a generic committee C_l (of size n).
- $M_{l,1}, \dots, M_{l,n}$ denotes the roles of the committee (of size n) responsible for the lth multiplication.
- $O_{l,1}, \dots, O_{l,n}$ denotes the roles of the committee (of size n) who holds the output of the lth multiplication. (If committee O_l is responsible for the next multiplication, it will be the same as committee M_{l+1}; or it can be the output committee.)
- $A_{l,1}, \dots, A_{l,h}$ and $B_{l,1}, \dots, B_{l,h}$ denote the roles of the two helper committees (each of size h) responsible for the generation of Beaver triples to aid in the round l multiplication.
- $Z_1, \dots, Z_h$ denotes the roles of a helper committee (of size h) responsible for generating a sharing of zero to help mask the computation output.

For simplicity, we assume that each committee only performs a single operation (whether it be decryption, Beaver triple preparation or multiplication). This can easily be parallelized so that each committee does a single *level* of operations.

Protocol Π

Prepare: *This step is run by a committee T of size n, the members of which do not require RACC's.*
To prepare the common reference strings to be used in multi-string NIZK proofs, each member T_i ($i \in [n]$) of the preparation committee T broadcasts
- $crs_{\mathsf{Share},i} \leftarrow \mathsf{mNIZK.Gen}(1^\kappa, \mathcal{R}_{\mathsf{Share}})$,
- $crs_{\mathsf{Dec},i} \leftarrow \mathsf{mNIZK.Gen}(1^\kappa, \mathcal{R}_{\mathsf{Dec}})$,
- $crs_{A,i} \leftarrow \mathsf{mNIZK.Gen}(1^\kappa, \mathcal{R}_{\mathsf{Beaver,A}})$,
- $crs_{B,i} \leftarrow \mathsf{mNIZK.Gen}(1^\kappa, \mathcal{R}_{\mathsf{Beaver,B}})$, and
- $crs_{\mathsf{MakeZero},i} \leftarrow \mathsf{mNIZK.Gen}(1^\kappa, \mathcal{R}_{\mathsf{MakeZero}})$.

Everyone can now locally compute
- $\overline{crs}_{\mathsf{Share}} := (crs_{\mathsf{Share},1}, \dots, crs_{\mathsf{Share},n})$,

- $\overline{crs}_{\mathsf{Dec}} := (crs_{\mathsf{Dec},1}, \ldots, crs_{\mathsf{Dec},n})$,
- $\overline{crs}_A := (crs_{A,1}, \ldots, crs_{A,n})$,
- $\overline{crs}_B := (crs_{B,1}, \ldots, crs_{B,n})$, and
- $\overline{crs}_{\mathsf{MakeZero}} := (crs_{\mathsf{MakeZero},1}, \ldots, crs_{\mathsf{MakeZero},n})$.

Input: *This step is run by an input role, which does not require RACC's.*
To provide input x to committee C, the jth input role does the following:
- Computes a shamir sharing of its secret input x as $(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x; \rho)$ with threshold t.
- Encrypts the shares to the committee members; i.e., for $i \in [n]$, encrypts share x_i as $\overline{x_i} \leftarrow \mathsf{Enc}(pk_{C_i}, x_i; \rho_i)$.
- Sets $\phi_{\mathsf{Share}} := \left(j, pk_{C_1}, \ldots, pk_{C_n}, \overline{x_1}, \ldots, \overline{x_n}\right)$ and $w_{\mathsf{Share}} := \left(x, \rho, \{\rho_i\}_{i \in [n]}\right)$. Computes the multi-string NIZK proof $\pi_{\mathsf{Share}} \leftarrow \mathsf{P}(\overline{crs}_{\mathsf{Share}}, \phi_{\mathsf{Share}}, w_{\mathsf{Share}})$ for the NP language $\mathcal{R}_{\mathsf{Share}}$.
- Broadcasts $(\pi_{\mathsf{Share}}, \overline{x_1}, \ldots, \overline{x_n})$.

All members $C_1, \ldots, C_n$ of committee C check the proof π_{Share}, and reject the input if the proof does not verify.

Decrypt: *This step is run by a committee C of size n, the members of which require RACC's.*
To open a value $(\overline{x_1}, \ldots, \overline{x_n})$, each member C_i of committee C does the following:
- Decrypts its share as $x_i \leftarrow \mathsf{Dec}(sk_{C_i}, \overline{x_i})$.
- Sets $\phi_{\mathsf{Dec},i} := \{pk_{C_i}, x_i, \overline{x_i}\}$ and $w_{\mathsf{Dec},i} := \{sk_{C_i}\}$. Computes the multi-string NIZK proof $\pi_{\mathsf{Dec},i} \leftarrow \mathsf{P}(\overline{crs}_{\mathsf{Dec}}, \phi_{\mathsf{Dec},i}, w_{\mathsf{Dec},i})$ for the NP language $\mathcal{R}_{\mathsf{Dec}}$.
- Broadcasts $(\pi_{\mathsf{Dec},i}, x_i)$.

Let $\mathcal{Q} \subseteq [n]$ denote the indices of the roles who provided a valid share (as determined by the verification of the multi-string NIZK proof). Anyone can then reconstruct x as $x \leftarrow \mathsf{Rec}(\{x_i\}_{i \in \mathcal{Q}})$.

Add: *This step is run by everyone and does not require RACC's.*
To add values $(\overline{x_1}, \ldots, \overline{x_n})$ and $(\overline{y_1}, \ldots, \overline{y_n})$ (both encrypted to the same committee C), everyone publicly computes element-wise homomorphic addition of the two vectors of ciphertexts as $\overline{z_i} := \mathsf{Eval}(pk_{C_i}, (\overline{x_i}, \overline{y_i}), (1, 1))$ for $i \in [n]$.

MakeBeaver: *This step is run by two helper committees A_l and B_l of size h each, the members of which do not require RACC's.*
To produce a Beaver triple for the lth multiplication, the members of the two helper committees A_l and B_l proceed as follows. (Multiplication then reduces to linear operations and decryptions, described below.) Note that the Beaver values must be shared to *two committees*: the committee M_l responsible for performing the multiplication, and the committee O_l who will hold the output. Committee O_l may then be asked to perform linear operations, another multiplication, or simply to decrypt the output.
- Each member $A_{l,j}$ of committee A_l does the following:
  - Picks a random value $a_{l,j}$.
  - Computes two sharings of $a_{l,j}$ as
    - $(a_{l,j,1}, \ldots, a_{l,j,n}) \leftarrow \mathsf{Share}(a_{l,j}; \rho_{l,j})$ and
    - $(a'_{l,j,1}, \ldots, a'_{l,j,n}) \leftarrow \mathsf{Share}(a_{l,j}; \rho'_{l,j})$

    with threshold t.
  - Encrypts the shares to the lth multiplication and output committees; i.e., for $i \in [n]$, she encrypts
    - $\overline{a_{l,j,i}} \leftarrow \mathsf{Enc}(pk_{M_{l,i}}, a_{l,j,i}; \rho_{l,j,i})$ and
    - $\overline{a_{l,j,i}}' \leftarrow \mathsf{Enc}(pk_{O_{l,i}}, a'_{l,j,i}; \rho'_{l,j,i})$.
  - Sets
    - $\phi_{A,l,j} := \left(l, j, \{pk_{M_{l,i}}\}_{i \in [n]}, \{pk_{O_{l,i}}\}_{i \in [n]}, \overline{a_{l,j,1}}, \ldots, \overline{a_{l,j,n}}, \overline{a_{l,j,1}}', \ldots, \overline{a_{l,j,n}}'\right)$ and
    - $w_{A,l,j} := \left(a_{l,j}, \rho_{l,j}, \rho'_{l,j}, \{\rho_{l,j,i}\}_{i \in [n]}, \{\rho'_{l,j,i}\}_{i \in [n]}\right)$.

    Computes the multi-string NIZK proof $\pi_{A,l,j} \leftarrow \mathsf{P}(\overline{crs}_A, \phi_{A,l,j}, w_{A,l,j})$ for the NP language $\mathcal{R}_{\mathsf{Beaver,A}}$.
  - Broadcasts $(\pi_{A,l,j}, \overline{a_{l,j,1}}, \ldots, \overline{a_{l,j,n}}, \overline{a_{l,j,1}}', \ldots, \overline{a_{l,j,n}}')$.
- Let $\mathcal{Q}_{A_l} \subseteq [h]$ denote the indices of the roles who provided a verifying proof.
  For each $i \in [n]$, anyone can then compute a sharing of $a_l = \sum_{j \in \mathcal{Q}_{A_l}} a_{l,j}$ encrypted to committee M_l and O_l using the homomorphisms of the encryption and sharing schemes. More formally, everyone computes
  - $\overline{a_{l,i}}$ as the homomorphic sum of the ciphertexts $\{\overline{a_{l,j,i}}\}_{j \in \mathcal{Q}_{A_l}}$ (i.e. $\overline{a_{l,i}} := \mathsf{Eval}(pk_{M_{l,i}}, \{\overline{a_{l,j,i}}\}_{j \in \mathcal{Q}_{A_l}}, (1, \ldots, 1))$), and

    - $\overline{a_{l,i}}'$ as the homomorphic sum of the ciphertexts $\{\overline{a_{l,j,i}}'\}_{j\in\mathcal{Q}_{A_l}}$ (i.e. $\overline{a_{l,i}}' := \mathsf{Eval}(pk_{O_{l,i}}, \{\overline{a_{l,j,i}}'\}_{j\in\mathcal{Q}_{A_l}}, (1,\ldots,1)))$.
  - Each member $B_{l,j}$ of committee B_l does the following:
    - Picks a random value $b_{l,j}$.
    - Computes two sharings of $b_{l,j}$ as
      * $(b_{l,j,1},\ldots,b_{l,j,n}) \leftarrow \mathsf{Share}(b_{l,j};\rho_{l,j})$ and
      * $(b'_{l,j,1},\ldots,b'_{l,j,n}) \leftarrow \mathsf{Share}(b_{l,j};\rho'_{l,j})$

      with threshold t.
    - Encrypts the shares to the lth multiplication and output committees; i.e., for $i\in[n]$, she encrypts
      * $\overline{b_{l,j,i}} \leftarrow \mathsf{Enc}(pk_{M_{l,i}}, b_{l,j,i};\rho_{l,j,i})$ and
      * $\overline{b_{l,j,i}}' \leftarrow \mathsf{Enc}(pk_{O_{l,i}}, b'_{l,j,i};\rho'_{l,j,i})$.
    - For each $i\in[n]$, homomorphically multiplies the ciphertext $\overline{a_{l,i}}'$ by the constant $b_{l,j}$ to get $\overline{c_{l,j,i}}'$ (i.e., computes $\overline{c_{l,j,i}}' := \mathsf{Eval}(pk_{O_{l,i}}, \overline{a_{l,i}}', b_{l,j})$).
    - Sets
      * $\phi_{B,l,j} := \big(l, j, \{pk_{M_{l,i}}\}_{i\in[n]}, \{pk_{O_{l,i}}\}_{i\in[n]}, \overline{b_{l,j,1}},\ldots,\overline{b_{l,j,n}}, \overline{b_{l,j,1}}',\ldots,\overline{b_{l,j,n}}', \overline{c_{l,j,1}}',\ldots,\overline{c_{l,j,n}}', \overline{a_{l,1}}',\ldots,\overline{a_{l,n}}'\big)$ and
      * $w_{B,l,j} := \big(b_{l,j}, \rho_{l,j}, \rho'_{l,j}, \{\rho_{l,j,i}\}_{i\in[n]}, \{\rho'_{l,j,i}\}_{i\in[n]}\big)$.

      Compute the multi-string NIZK proof $\pi_{B,l,j} \leftarrow \mathsf{P}(\overline{crs}_B, \phi_{B,l,j}, w_{B,l,j})$ for the NP language $\mathcal{R}_{\mathsf{Beaver,B}}$.
    - Broadcasts $(\pi_{B,l,j}, \overline{b_{l,j,1}},\ldots,\overline{b_{l,j,n}}, \overline{b_{l,j,1}}',\ldots,\overline{b_{l,j,n}}', \overline{c_{l,j,1}}',\ldots,\overline{c_{l,j,n}}')$.
  - Let $\mathcal{Q}_{B_l}\subseteq[h]$ denote the indices of the roles who provided a verifying proof.
    For each $i\in[n]$, anyone can then compute a sharing of $b_l = \Sigma_{j\in\mathcal{Q}_{B_l}} b_{l,j}$ encrypted to committee M_l and O_l using the homomorphisms of the encryption and sharing schemes. Further, everyone also computes the sharing of $c_l = a_l b_l$ encrypted to committee O_l. More formally, everyone computes
    - $\overline{b_{l,i}}$ as the homomorphic sum of the ciphertexts $\{\overline{b_{l,j,i}}\}_{j\in\mathcal{Q}_{B_l}}$ (i.e. $\overline{b_{l,i}} := \mathsf{Eval}(pk_{M_{l,i}}, \{\overline{b_{l,j,i}}\}_{j\in\mathcal{Q}_{B_l}}, (1,\ldots,1)))$,
    - $\overline{b_{l,i}}'$ as the homomorphic sum of the ciphertexts $\{\overline{b_{l,j,i}}'\}_{j\in\mathcal{Q}_{B_l}}$ (i.e. $\overline{b_{l,i}}' := \mathsf{Eval}(pk_{O_{l,i}}, \{\overline{b_{l,j,i}}'\}_{j\in\mathcal{Q}_{B_l}}, (1,\ldots,1)))$, and
    - $\overline{c_{l,i}}'$ as the homomorphic sum of the ciphertexts $\{\overline{c_{l,j,i}}'\}_{j\in\mathcal{Q}_{B_l}}$ i.e. $(\overline{c_{l,i}}' := \mathsf{Eval}(pk_{O_{l,i}}, \{\overline{c_{l,j,i}}'\}_{j\in\mathcal{Q}_{B_l}}, (1,\ldots,1)))$.

Note that the entirety of Beaver triple generation can be carried out without the helper committees needing to receive any private messages. Additionally, note that Beaver triple generation committees can have a dishonest majority, and can therefore be smaller ($h < n$).

Mult: *This step is run by committee M_l of size n, the members of which require RACC's.*
To multiply values
  - $(\overline{x_{l,1}},\ldots,\overline{x_{l,n}})$ (representing x_l) and
  - $(\overline{y_{l,1}},\ldots,\overline{y_{l,n}})$ (representing y_l)

(encrypted to committee M_l) using the Beaver triple
  - $(\overline{a_{l,1}},\ldots,\overline{a_{l,n}})$, $(\overline{a_{l,1}}',\ldots,\overline{a_{l,n}}')$,
  - $(\overline{b_{l,1}},\ldots,\overline{b_{l,n}})$, $(\overline{b_{l,1}}',\ldots,\overline{b_{l,n}}')$
    (encrypted to committees M_l and O_l, respectively) and
  - $(\overline{c_{l,1}}',\ldots,\overline{c_{l,n}}')$, (encrypted to committee O_l),

We do the following:
  - Everyone publicly computes a sharing of $\epsilon_l = a_l - x_l$ and $\delta_l = b_l - y_l$ encrypted to committee M_l using the homomorphisms of the encryption and sharing schemes. More formally, everyone computes
    - the homomorphic difference of the ciphertexts $\overline{a_{l,i}}$ and $\overline{x_{l,i}}$ to get $\overline{\epsilon_{l,i}}$ for $i\in[n]$ (i.e. $\overline{\epsilon_{l,i}} := \mathsf{Eval}(pk_{M_{l,i}}, (\overline{a_{l,i}}, \overline{x_{l,i}}), (1,-1)))$, and
    - the homomorphic difference of the ciphertexts $\overline{b_{l,i}}$ and $\overline{y_{l,i}}$ to get $\overline{\delta_{l,i}}$ for $i\in[n]$ (i.e. $\overline{\delta_{l,i}} := \mathsf{Eval}(pk_{M_{l,i}}, (\overline{b_{l,i}}, \overline{y_{l,i}}), (1,-1)))$.
  - Each member $C_{l,i}$ of committee M_l opens the values $\overline{\epsilon_{l,i}}$ and $\overline{\delta_{l,i}}$ to reconstruct ϵ_l and δ_l (as described above in $\mathsf{Decrypt}$).
  - Everyone publicly computes a sharing of $x_l y_l = c_l - \epsilon_l b_l - \delta_l a_l + \epsilon_l\delta_l$ encrypted to committee O_l using the homomorphisms of the encryption and sharing schemes. More formally, for each $i\in[n]$, everyone computes
    - the ciphertext corresponding to plaintext 1 as $\overline{1} \leftarrow \mathsf{Enc}(pk_{O_{l,i}}, 1; \rho'')$ using some default randomness ρ'', and

- the ciphertext corresponding to each share of $z_l = x_l y_l$ as $\overline{z_{l,i}}' := \mathsf{Eval}(pk_{O_{l,i}}, (\overline{c_{l,i}}', \overline{b_{l,i}}', \overline{a_{l,i}}', \overline{1}), (1, -\epsilon_l, -\delta_l, \epsilon_l\delta_l))$.

MakeZero: *This step is run by a helper committee Z of size h, the members of which do not require RACC's.*
To produce a sharing of zero held by the output committee C, each member Z_j of committee Z does the following:
- Computes a random sharing of 0 as $(0_{j,1}, \ldots, 0_{j,n}) \leftarrow \mathsf{Share}(0; \rho_j)$ with threshold t.
- Encrypts the shares to the members of committee C; i.e., for $i \in [n]$, she encrypts $\overline{0_{j,i}} \leftarrow \mathsf{Enc}(pk_{C_i}, 0_{j,i}; \rho_{j,i})$.
- Sets $\phi_{\mathsf{MakeZero}} := (j, pk_{C_1}, \ldots, pk_{C_n}, \overline{0_{j,1}}, \ldots, \overline{0_{j,n}})$ and $w_{\mathsf{MakeZero}} := (\rho, \{\rho_{j,i}\}_{i\in[n]})$. Computes the multi-string NIZK proof $\pi_{\mathsf{MakeZero}} \leftarrow \mathsf{P}(\overline{crs}_{\mathsf{MakeZero}}, \phi_{\mathsf{MakeZero}}, w_{\mathsf{MakeZero}})$ for the NP language $\mathcal{R}_{\mathsf{MakeZero}}$.
- Broadcasts $(\pi_{\mathsf{MakeZero}}, \overline{0_{j,1}}, \ldots, \overline{0_{j,n}})$.

Let $\mathcal{Q}_Z \subseteq [h]$ denote the indices of the roles who provided a verifying proof. For each $i \in [n]$, anyone can then compute the final shares of 0 by using the homomorphisms of the schemes. More formally, everyone computes $\overline{0_i}$ as the homomorphic sum of the ciphertexts $\{\overline{0_{j,i}}\}_{j\in\mathcal{Q}_Z}$ (i.e. $\overline{0_i} := \mathsf{Eval}(pk_{C_i}, \{\overline{0_{j,i}}\}_{j\in\mathcal{Q}_Z}, (1, \ldots, 1))$).

Output: *This step is run by the output committee C of size n, the members of which require RACC's.*
To output a value $(\overline{\mathsf{out}_1}, \ldots, \overline{\mathsf{out}_n})$ (representing out), the committee C uses the shares of zero $(\overline{0_1}, \ldots, \overline{0_n})$ produced by the helper committee Z. The shares of zero are necessary to randomize the output.
Anyone can compute the the sum of the encrypted output shares and encrypted shares of 0. More formally, everyone computes $\overline{\mathsf{out}_i}'$ as the homomorphic sum of $\overline{\mathsf{out}_i}$ and $\overline{0_i}$ (i.e. $\overline{\mathsf{out}_i}' := \mathsf{Eval}(pk_{C_i}, (\overline{\mathsf{out}_i}, \overline{0_i}), (1, 1))$).
The committee C then calls Decrypt on $(\overline{\mathsf{out}_1}', \ldots, \overline{\mathsf{out}_n}')$.

3.2 Proof of Security

We start by analyzing the correctness of an all-honest execution of the protocol.

Lemma 1 (Correctness). *The protocol Π on inputs $(x_1, \ldots, x_m)$ produces output value $z = f(x_1, \ldots, x_m)$ when all roles are honest.*

Proof. Correctness follows from the evaluation of each gate producing a sharing of the appropriate gate output. If this is the case the output may be reconstructed from the sharing associated with the final gate by perfect correctness of the secret sharing scheme. Proofs produced honestly will be accepting due to completeness of the multi-string NIZK proof system.

Input The role giving input distributes a sharing of its input value x by construction.

Add Perfect correctness of the Shamir secret sharing and the linearly homomorphic encryption scheme, and the compatibility of their homomorphisms, ensures the output is a sharing of $z = x + y$.

Mult The values produced by MakeBeaver are valid sharings of a, b and c, where $c = ab$. It follows by inspection that $z = c - \epsilon b - \delta a + \epsilon\delta = c - (a - x)b - (b - y)a + (a - x)(b - y) = xy$.

We will now prove the security of our YOSO-LHE protocol.

Theorem 1. *For any m-input function $f : (\{0,1\}^*)^m \rightarrow \{0,1\}^*$, the protocol Π described above YOSO-realizes f (with guaranteed output delivery) as long as more than half the members of each size-n committee are honest, and at least one member of each size-h committee is honest.*

The fact that the YOSO property is preserved is evident from the fact that every part speaks at most once. It remains to prove UC-security.

Proof. We start by defining a simulator $\mathcal{S}$ which may be composed with any PPT real-world adversary $\mathcal{A}$ to produce an ideal world adversary $\mathcal{S}'$ such that for every PPT environment $\mathcal{E}$, it holds that $REAL_{\Pi,\mathcal{A},\mathcal{E}}(x)$ and $IDEAL_{\mathcal{F}_f,\mathcal{S},\mathcal{E}}(x)$ are indistinguishable. We prove indistinguishability through a series of hybrids, each indistinguishable from the last, starting in the real world and arriving in the ideal world with our complete simulator.

We define the simulator $\mathcal{S}$ below. We denote the set of honest and corrupt roles as $\mathcal{H}$ and $\mathcal{I}$ respectively; we let $\mathcal{H}_C$ and $\mathcal{I}_C$ represent the honest and corrupt roles within a committee C.

Simulator $\mathcal{S}$

Prepare:
1. *Simulating the common reference strings.* On behalf of each honest role $i \in \mathcal{H}_T$, $\mathcal{S}$ computes
   - $(crs_{\mathsf{Share},i}, \tau_{\mathsf{Share},i}, \xi_{\mathsf{Share},i}) \leftarrow \mathsf{SE}_1(1^\kappa, \mathcal{R}_{\mathsf{Share}})$,
   - $(crs_{\mathsf{Dec},i}, \tau_{\mathsf{Dec},i}, \xi_{\mathsf{Dec},i}) \leftarrow \mathsf{SE}_1(1^\kappa, \mathcal{R}_{\mathsf{Dec}})$,
   - $(crs_{A,i}, \tau_{A,i}, \xi_{A,i}) \leftarrow \mathsf{SE}_1(1^\kappa, \mathcal{R}_{\mathsf{Beaver,A}})$,
   - $(crs_{B,i}, \tau_{B,i}, \xi_{B,i}) \leftarrow \mathsf{SE}_1(1^\kappa, \mathcal{R}_{\mathsf{Beaver,B}})$, and
   - $(crs_{\mathsf{MakeZero},i}, \tau_{\mathsf{MakeZero},i}, \xi_{\mathsf{MakeZero},i}) \leftarrow \mathsf{SE}_1(1^\kappa, \mathcal{R}_{\mathsf{MakeZero}})$,

   and broadcasts $(crs_{\mathsf{Share},i}, crs_{\mathsf{Dec},i}, crs_{A,i}, crs_{B,i}, crs_{\mathsf{MakeZero},i})$.
2. Let $(crs_{\mathsf{Share},j}, crs_{\mathsf{Dec},j}, crs_{A,j}, crs_{B,j}, crs_{\mathsf{MakeZero},j})$ denote the crs's received from corrupt role $j \in \mathcal{I}_T$.
3. Let $(\overline{crs}_{\mathsf{Share}}, \overline{crs}_{\mathsf{Dec}}, \overline{crs}_A, \overline{crs}_B, \overline{crs}_{\mathsf{MakeZero}})$ denote the vectors of n common reference strings.
4. Let $(\overline{\tau}_{\mathsf{Share}}, \overline{\tau}_{\mathsf{Dec}}, \overline{\tau}_A, \overline{\tau}_B, \overline{\tau}_{\mathsf{MakeZero}})$ denote the vectors of simulation trapdoors (comprising of the relevant τ's of roles in $\mathcal{H}_T$).
5. Let $(\overline{\xi}_{\mathsf{Share}}, \overline{\xi}_{\mathsf{Dec}}, \overline{\xi}_A, \overline{\xi}_B, \overline{\xi}_{\mathsf{MakeZero}})$ denote the vectors of extraction trapdoors (comprising of the relevant ξ's of roles in $\mathcal{H}_T$).

Input:
1. *Simulating encryptions.* On behalf of the jth honest input role, compute the encryptions $\overline{x_{j,1}}, \ldots, \overline{x_{j,n}}$ such that $\overline{x_{j,i}}$ corresponds to encryptions of 0 when $i \in \mathcal{H}_C$ and encryptions of random shares when $i \in \mathcal{I}_C$.
2. *Simulating the proofs.* For jth honest input role, set $\phi_{\mathsf{Share},j} = (j, pk_{C_1}, \ldots, pk_{C_n}, \overline{x_{j,1}}, \ldots, \overline{x_{j,n}})$ and compute the simulated multi-string NIZK proof as $\pi_{\mathsf{Share},j} \leftarrow \mathsf{S}_2(\overline{crs}_{\mathsf{Share}}, \phi_{\mathsf{Share},j}, \overline{\tau}_{\mathsf{Share}})$ for the NP language $\mathcal{R}_{\mathsf{Share}}$. Broadcast $(\pi_{\mathsf{Share},j}, \overline{x_{j,1}}, \ldots, \overline{x_{j,n}})$.
3. *Extraction of values held by corrupt roles.* Let $\mathcal{I}'$ denote corrupt input roles who provide a verifying proof. For each $j \in \mathcal{I}'$, run $w_j \leftarrow \mathsf{E}_2(\overline{crs}_{\mathsf{Share}}, \phi_{\mathsf{Share},j}, \pi_{\mathsf{Share},j}, \overline{\xi}_{\mathsf{Share}})$ where $\phi_{\mathsf{Share},j}$ and $\pi_{\mathsf{Share},j}$ were received from the adversary. If $(\phi_{\mathsf{Share},j}, w_j) \notin \mathcal{R}_{\mathsf{Share}}$ (i.e. the extraction of the witness fails), then abort. Otherwise, $\mathcal{S}$ learns x_j and the randomness used by the jth corrupt role (for Shamir sharing its input). This is used to deduce the input shares held by corrupt roles in C. Note that the shares held by corrupt roles in C corresponding to honest inputs are already known to $\mathcal{S}$ (as she defined them).
4. *Invoking the ideal functionality.* Invoke $\mathcal{F}_f$ with x_j on behalf of roles in $\mathcal{I}'$ (the corrupt input roles who provided a verifying proof) and default inputs on behalf of the remaining corrupt input roles. Let out denote the output obtained from $\mathcal{F}_f$.

Decrypt:
1. If the value being decrypted corresponds to the ϵ or δ values during computation of multiplication gate, set plaintext x as a random value. Otherwise, this value corresponds to decryption of the final output and the plaintext x is set to out.
2. *Simulating the decryption shares.* Let $\{x_i\}_{i \in \mathcal{I}_C}$ denote the shares held by corrupt roles in C (which the simulator knows because these have been deduced at every computation

stage). Compute the shares on behalf of honest roles such that they would be consistent with x as follows: Compute $\{x'_i\}_{i\in\mathcal{H}_C} \leftarrow \mathsf{SH.SimShare}(\{x_i\}_{i\in\mathcal{I}_C}, x)$.

3. *Simulating the proofs.* For each $i \in \mathcal{H}_C$, set $\phi_{\mathsf{Dec},i} = \{pk_{C_i}, x'_i, \overline{x_i}\}$. Compute the simulated multi-string NIZK proof as $\pi_{\mathsf{Dec},i} \leftarrow \mathsf{S}_2(\overline{crs}_{\mathsf{Dec}}, \phi_{\mathsf{Dec},i}, \overline{\tau}_{\mathsf{Dec}})$ for the NP language $\mathcal{R}_{\mathsf{Dec}}$. Broadcast $(\pi_{\mathsf{Dec},i}, x'_i)$.

Add: $\mathcal{S}$ uses the values learned earlier and homomorphism of secret sharing to deduce the shares of the output of the addition gate, i.e., $z = x + y$ (where x and y denote the inputs to the addition gate) held by corrupt roles in C.

MakeBeaver:

1. $\mathcal{S}$ does the following on behalf of honest helper $j \in \mathcal{H}_{A_l}$:
   - *Simulating encryptions.* Compute encryptions $\overline{a_{l,j,1}}, \ldots, \overline{a_{l,j,n}}$, such that $\overline{a_{l,j,i}}$ corresponds to encryptions of 0 when $i \in \mathcal{H}_{M_l}$ and encryptions of random shares when $i \in \mathcal{I}_{M_l}$.
   - *Simulating encryptions.* Compute encryptions $\overline{a_{l,j,1}}', \ldots, \overline{a_{l,j,n}}'$, such that $\overline{a_{l,j,i}}'$ corresponds to encryptions of 0 when $i \in \mathcal{H}_{O_l}$ and encryptions of random shares when $i \in \mathcal{I}_{O_l}$.
   - *Simulating the proofs.* Set $\phi_{A,l,j} := \big(l, j, \{pk_{M_{l,i}}\}_{i\in[n]}, \{pk_{O_{l,i}}\}_{i\in[n]}, \overline{a_{l,j,1}}, \ldots, \overline{a_{l,j,n}}, \overline{a_{l,j,1}}', \ldots, \overline{a_{l,j,n}}'\big)$ and compute the simulated multi-string NIZK proof as $\pi_{A,l,j} \leftarrow \mathsf{S}_2(\overline{crs}_A, \phi_{A,l,j}, \overline{\tau}_A)$ for the NP language $\mathcal{R}_{\mathsf{Beaver,A}}$. Broadcast $(\pi_{A,l,j}, \overline{a_{l,j,1}}, \ldots, \overline{a_{l,j,n}}, \overline{a_{l,j,1}}', \ldots, \overline{a_{l,j,n}}')$.
2. *Extraction of values held by corrupt roles.* Let $\mathcal{I}' = \mathcal{I}_{A_l} \cap \mathcal{Q}_{A_l}$ denote the set of corrupt roles in A_l who provide a verifying proof. Then for each $j \in \mathcal{I}'$, run $w_{A,l,j} \leftarrow \mathsf{E}_2(\overline{crs}_A, \phi_{A,l,j}, \pi_{A,l,j}, \overline{\xi}_A)$. If $(\phi_{A,l,j}, w_{A,l,j}) \notin \mathcal{R}_{\mathsf{Beaver,A}}$ (i.e. the extraction of the witness fails), then abort. Otherwise, $\mathcal{S}$ can use the witness to deduce the shares of a_l held by the corrupt roles in M_l and O_l.
3. $\mathcal{S}$ does the following on behalf of honest helper $j \in \mathcal{H}_{B_l}$:
   - *Simulating encryptions.* Compute encryptions $\overline{b_{l,j,1}}, \ldots, \overline{b_{l,j,n}}$, such that $\overline{b_{l,j,i}}$ corresponds to encryptions of 0 when $i \in \mathcal{H}_{M_l}$ and encryptions of random shares when $i \in \mathcal{I}_{M_l}$.
   - *Simulating encryptions.* Compute encryptions $\overline{b_{l,j,1}}', \ldots, \overline{b_{l,j,n}}'$, such that $\overline{b_{l,j,i}}'$ corresponds to encryptions of 0 when $i \in \mathcal{H}_{O_l}$ and encryptions of random shares when $i \in \mathcal{I}_{O_l}$.
   - *Simulating encryptions.* Compute encryptions $\overline{c_{l,j,1}}', \ldots, \overline{c_{l,j,n}}'$, such that $\overline{c_{l,j,i}}'$ corresponds to encryptions of 0 when $i \in \mathcal{H}_{O_l}$ and encryptions of random shares when $i \in \mathcal{I}_{O_l}$.
   - *Simulating the proofs.* Set $\phi_{B,l,j} := \big(l, j, \{pk_{M_{l,i}}\}_{i\in[n]}, \{pk_{O_{l,i}}\}_{i\in[n]}, \overline{b_{l,j,1}}, \ldots, \overline{b_{l,j,n}}, \overline{b_{l,j,1}}', \ldots, \overline{b_{l,j,n}}', \overline{c_{l,j,1}}', \ldots, \overline{c_{l,j,n}}', \overline{a_{l,1}}', \ldots, \overline{a_{l,n}}'\big)$.
     Compute the simulated multi-string NIZK proof as $\pi_{B,l,j} \leftarrow \mathsf{S}_2(\overline{crs}_B, \phi_{B,l,j}, \overline{\tau}_B)$ for the NP language $\mathcal{R}_{\mathsf{Beaver,B}}$. Broadcast $(\pi_{B,l,j}, \overline{b_{l,j,1}}, \ldots, \overline{b_{l,j,n}}, \overline{b_{l,j,1}}', \ldots, \overline{b_{l,j,n}}' \overline{c_{l,j,1}}', \ldots, \overline{c_{l,j,n}}')$.
4. *Extraction of values held by corrupt roles.* Let $\mathcal{I}' = \mathcal{I}_{B_l} \cap \mathcal{Q}_{B_l}$ denote the corrupt roles in B_l who provide a verifying proof. Then for each $j \in \mathcal{I}'$, run $w_{B,l,j} \leftarrow \mathsf{E}_2(\overline{crs}_B, \phi_{B,l,j}, \pi_{B,l,j}, \overline{\xi}_B)$. If $(\phi_{B,l,j}, w_{B,l,j}) \notin \mathcal{R}_{\mathsf{Beaver,B}}$ (i.e. the extraction of the witness fails), then abort. Else, $\mathcal{S}$ can use the witness to deduce the shares of b_l held by the corrupt roles in M_l and O_l and the shares of c_l held by corrupt roles in O_l.

Mult:

1. Deduce the shares of ϵ_l and δ_l held by corrupt roles in M_l using the shares learned earlier and homomorphism of the secret sharing.
2. Execute the simulation steps in Decrypt to open ϵ_l and δ_l on behalf of honest roles in M_l.
3. Deduce the shares of the output of the multiplication gate, i.e., $x_l y_l$ (where x_l and y_l denote the inputs to the multiplication gate) held by corrupt roles in O_l using the homomorphism of secret sharing (and the values learned earlier).

MakeZero:

1. *Simulating encryptions.* On behalf of the $j \in \mathcal{H}_Z$, compute the encryptions $\overline{0_{j,1}}, \ldots, \overline{0_{j,n}}$ such that $\overline{0_{j,i}}$ corresponds to encryptions of 0 when $i \in \mathcal{H}_C$ and encryptions of random shares when $i \in \mathcal{I}_C$.
2. *Simulating the proofs.* On behalf of the $j \in \mathcal{H}_Z$, set $\phi_{\mathsf{MakeZero},j} = \big(j, pk_{C_1}, \ldots, pk_{C_n}, \overline{0_{j,1}}, \ldots, \overline{0_{j,n}}\big)$ and compute the simulated multi-string NIZK proof as $\pi_{\mathsf{MakeZero},j} \leftarrow \mathsf{S}_2(\overline{crs}_{\mathsf{MakeZero}}, \phi_{\mathsf{MakeZero},j}, \overline{\tau}_{\mathsf{MakeZero}})$ for the NP language $\mathcal{R}_{\mathsf{MakeZero}}$. Broadcast $(\pi_{\mathsf{MakeZero},j}, \overline{0_{j,1}}, \ldots, \overline{0_{j,n}})$.
3. *Extraction of values held by corrupt roles.* Let $\mathcal{I}' = \mathcal{I}_Z \cap \mathcal{Q}_Z$ denote the corrupt roles in Z who provide a verifying proof. Then for each $j \in \mathcal{I}'$, run

$w_{\mathsf{MakeZero},j} \leftarrow \mathsf{E}_2(\overline{crs}_{\mathsf{MakeZero}}, \phi_{\mathsf{MakeZero},j}, \pi_{\mathsf{Share},j}, \bar{\xi}_{\mathsf{MakeZero}})$ where $\phi_{\mathsf{MakeZero},j}$ and $\pi_{\mathsf{MakeZero},j}$ was received from the adversary. If $(\phi_{\mathsf{MakeZero},j}, w_{\mathsf{MakeZero},j}) \notin \mathcal{R}_{\mathsf{MakeZero}}$ (i.e. the extraction of the witness fails), then abort. Otherwise, $\mathcal{S}$ uses the witness to learn the shares of zero held by the corrupt roles in the output committee C.

Output: Using the values learned earlier (including the shares of zero held by the corrupt roles in output committee C) and the homomorphism of secret sharing, $\mathcal{S}$ deduces the shares of the output (which are encrypted in $\overline{\mathsf{out}_i}$, for $i \in \mathcal{I}_C$) held by the corrupt roles in the output committee. Finally, $\mathcal{S}$ executes the simulation steps in Decrypt to open the final output out.

We describe a series of hybrid simulators allowing us to arrive at the full simulator described above. The final simulator does not require access to the inputs of honest roles, relying only on the ideal functionality.

Real: The simulator does everything as in the real protocol (in particular, it uses honest roles' real inputs).
Hybrid 1 (CRS Simulation): The simulator uses the multi-string NIZK simulator SE_1 to generate the common reference string for each of the honest roles in the Prepare committee. This is indistinguishable from real case by reference string indistinguishability.
Hybrid 2 (Proof Simulation): The simulator uses the simulation trapdoors obtained when producing reference strings for the honest roles to simulate all honest proofs. The honest majority of the Prepare committee and simulation indistinguishability mean that this hybrid is indistinguishable from the previous hybrid.
Hybrid 3 (Witness Extraction): In this hybrid the simulator extracts values provided by corrupt roles. The adversary provides values in three steps: Input, MakeBeaver and MakeZero. The simulator extracts witnesses to find the shares and input values provided by the corrupt roles, aborting if extraction fails. By simulation-extractability, extraction of a valid witness only fails with negligible probability, ensuring this hybrid is indistinguishable from the previous one.
There are several things to note here:

1. Due to the perfect correctness of the encryption scheme, there exists a unique value and set of shares that will be contained in any valid witness.
2. Due to the statement for $\mathcal{R}_{\mathsf{Share}}$ and $\mathcal{R}_{\mathsf{MakeZero}}$ including a role index, corrupt roles cannot reuse honest roles' proofs. This is also the case for $\mathcal{R}_{\mathsf{Beaver,A}}$ and $\mathcal{R}_{\mathsf{Beaver,B}}$, where the round number is also needed for disambiguation. For $\mathcal{R}_{\mathsf{Dec}}$, a roles' public key ins included in the statement, serving the same purpose.

At this point, the simulator no longer needs access to the honest roles' secret keys, since all shares are either produced by the simulator herself (on behalf of other honest roles), or are recovered from corrupt roles' witnesses. The simulator computes the shares on each wire of the circuit using the homomorphism of the secret sharing scheme.
The simulator gives the extracted corrupt roles' input values (together with the honest roles' input values) to the ideal functionality to get the output

$z \leftarrow \mathcal{F}_f(x_1, \ldots, x_m)$, which will later be used to replace the protocol output. Note input values without a verifying proof are replaced by default values, as in the protocol.

Hybrid 4 (Encrypt 0 to Honest Roles): The simulator replaces all encryptions between two honest roles with encryptions of 0. This is indistinguishable from the previous hybrid as an adversary with non-negligible distinguishing advantage can be reduced to an adversary breaking the to semantic security of our LHE scheme. This reduction is made possible by the fact that the simulator does not require access to secret keys for honest roles. During Decrypt the honest roles still open to the correct shares contained in their ciphertext, but they can do so because the simulator knows the shares without needing to decrypt (from extraction and linear computation).

Hybrid 5 (Abort if Adversary Decrypts Shares Incorrectly): In this hybrid the simulator aborts if the adversary is able to open a corrupt role's ciphertext during Decrypt to something other than the share the simulator expects (as a result of extraction and linear computation). In the following hybrids the corrupt roles with verifying proofs will only decrypt the shares expected by the simulator. The simulation soundness of the $\pi_{\mathsf{Dec},i}$ proofs, the perfect correctness of both the LHE and secret sharing scheme and the key binding property of the LHE scheme ensure the simulator only aborts with negligible probability. (The KeyMatches algorithm forces the adversary to use a secret key which results in correct decryption.) Thus, this hybrid is indistinguishable from the previous one. Correctness of the protocol now follows from correctness in the honest setting and the honest majority ensuring reconstruction is always possible.

Hybrid 6 (Simulate Honest Roles' Output Shares): Let $\mathcal{I}_C$ and $\mathcal{H}_C$ denote the set of corrupt and honest roles, respectively, in the final output committee C. The simulator knows the ideal functionality's output out since Hybrid 3, so she can replace the honest shares in the final call to Decrypt; i.e., she now computes the honest roles' shares as $\{\mathsf{out}'_{l,i}\}_{i \in \mathcal{H}_C} \leftarrow$ $\mathsf{SimShare}(\{\mathsf{out}'_{l,i}\}_{i \in \mathcal{I}_C}, \mathsf{out})$.
This is indistinguishable from the previous hybrid since we added a random sharing of 0 produced by the MakeZero committee to the sharing of out. This results in a random sharing of the same value. An adversary distinguishing this hybrid from the previous can be used to break the share simulatability property of the secret sharing scheme. This is possible since the honest members of the committee that runs MakeZero has not sent any messages which are dependent on the shares it produces for honest roles.

Hybrid 7 (Pick ϵ, δ Randomly, Simulate Honest Roles' Shares): For each Decrypt which is part of a Mult, the simulator (a) picks the decrypted value (ϵ or δ) randomly, and (b) simulates the honest shares of that value; that is, $\{\epsilon'_{l,i}\}_{i \in \mathcal{H}_{M_l}} \leftarrow \mathsf{SimShare}(\{\epsilon_{l,i}\}_{i \in \mathcal{I}_{M_l}}, \epsilon)$ and $\{\delta'_{l,i}\}_{i \in \mathcal{H}_{M_l}} \leftarrow \mathsf{SimShare}(\{\delta_{l,i}\}_{i \in \mathcal{I}_{M_l}}, \delta)$.
Two facts ensure that this hybrid is indistinguishable from the previous one. The first is the presence of at least one honest role in each of the A and B committees in any MakeBeaver invocation. The honest role i in A has not sent any messages which depend on the value $a_{l,i}$ (all shares this role

sent were either random or 0); similarly, the honest role j in B has not sent any messages depending on $b_{l,j}$ and $c_{l,j}$. Thus, the view of the adversary is independent of the values of a_l, b_l and c_l, except when she sees ϵ_l and δ_l; as long as a_l and b_l are uniform, so are ϵ_l and δ_l, so we can pick them at random directly.
Simulating the remaining shares will be indistinguishable from the real derivation of those shares by share simulatability.
Hybrid 8 (Replace Corrupt Roles' Shares With Random): During the Input operation, instead of giving corrupt roles real secret shares of honest roles' inputs x, give them random shares. Because honest roles' shares have already been replaced with 0 in each ciphertext and with simulated shares in each decryption, the remaining shares (of which there are fewer than t) are indistinguishable from random shares by the privacy property of Shamir secret sharing. At this point, the simulator no longer requries access to the inputs of honest roles.

References

BCP03. Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 37–54. Springer, Heidelberg, November / December 2003.

BGG$^+$20. 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.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract) (informal contribution). In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, page 462. Springer, Heidelberg, August 1988.

CDN01. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Heidelberg, May 2001.

CGG$^+$21. Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: Secure multiparty computation with dynamic participants. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 94–123, Virtual Event, August 2021. Springer, Heidelberg.

CL15. Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. Cryptology ePrint Archive, Report 2015/047, 2015. `https://eprint.iacr.org/2015/047`.

GHK$^+$21. 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.

GHM$^+$21. Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. Random-index PIR with applications to large-scale secure MPC. In *Theory of Cryptography Conference*, 2021.

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

GO07. Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 323–341. Springer, Heidelberg, August 2007.

LJA$^+$18. Andrei Lapets, Frederick Jansen, Kinan Dak Albab, Rawane Issa, Lucy Qin, Mayank Varia, and Azer Bestavros. Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, COMPASS '18, New York, NY, USA, 2018. Association for Computing Machinery.

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

sod19. SODA: Scalable oblivious data analytics. `https://soda-project.eu/`, 2019.

VCAZ$^+$18. Meilof Veeningen, Supriyo Chatterjea, Horváth Anna Zsófia, Gerald Spindler, Eric Boersma, Peter van der Spek, Onno van der Galiën, Job Gutteling, Wessel Kraaij, and Thijs Veugen. Enabling analytics on sensitive medical data with secure multi-party computation. *Stud Health Technol Inform*, 2018.

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

A Tools for YOSO-LHE

We recap the tools we need in order to build YOSO-LHE. We make use of linearly homomorphic threshold secret sharing (as described in Section A.1), linearly homomorphic public key encryption (as described in Section A.2), and multi-string NIZK proofs (as described in Section A.3).

A.1 Linearly Homomorphic Secret Sharing

A t-out-of-n secret sharing scheme allows a party to "split" a secret into n shares in a field $\mathbb{F}$ that can be distributed among different parties. To reconstruct the original secret x at least $t+1$ shares need to be used. Such a secret sharing scheme is linearly homomorphic if it allows parties to locally evaluate linear functions on shared values.

Syntax. A t-out-of-n linearly homomorphic secret sharing scheme has the following algorithms:

$\mathsf{Share}(x;\rho) \to (s_1,\ldots,s_n)$**:** An algorithm that, given a secret x, outputs a set of n shares in a finite field $\mathbb{F}$.
$\mathsf{Rec}(\{s_i\}_{i\in S\subseteq[n],|S|>t}) \to x$**:** An algorithm that, given a vector of at least $t+1$ shares, outputs the secret x.
$\mathsf{Eval}((s_1,\ldots,s_m),(c_1,\ldots,c_m)) \to s$**:** An algorithm that, given some party i's shares $s_1,\ldots,s_m$ of secrets $x_1,\ldots,x_m$ as well as coefficients $c_1,\ldots,c_m$, outputs a share s of $\sum_{j=1}^m c_j x_j$ in the finite field $\mathbb{F}$.
$\mathsf{SimShare}(\{s_i\}_{i\in S,|S|\le t}, x) \to \{s_i''\}_{i\in[n]\setminus S}$**:** A simulation algorithm that, given shares belonging to corrupt parties and a target value x, simulates the shares belonging to honest parties that causes Rec to output the desired value.

Properties. We require the following properties of a linearly homomorphic t-out-of-n secret sharing scheme:

Perfect Correctness. The perfect correctness property requires that the shares of a secret x should always reconstruct to x. More formally, a secret sharing scheme is *perfectly correct* if for any secret x, for any subset $S \subseteq [n], |S| > t$,

$$\Pr\left[x = x' \;\middle|\; \begin{array}{l} (s_1,\ldots,s_n) \leftarrow \mathsf{Share}(x) \\ x' \leftarrow \mathsf{Rec}(\{s_i\}_{i\in S}) \end{array} \right] = 1,$$

where the probability is taken over the random coins of Share.
Furthermore, correctness should hold even when shares are a result of an evaluation. More generally, the perfect correctness of a linearly homomorphic t-out-of-n secret sharing scheme requires that for any set of secrets $x_1,\ldots,x_m$, any set of coefficients $c_1,\ldots,c_m$, for any subset $S \subseteq [n], |S| > t$,

$$\Pr\left[x' = \sum_{j=1}^m c_j x_j \;\middle|\; \begin{array}{l} (s_1^j,\ldots,s_n^j) \leftarrow \mathsf{Share}(x_j)\ \forall j \in [m] \\ s_i \leftarrow \mathsf{Eval}((s_i^1,\ldots,s_i^m),(c_1,\ldots,c_m))\ \forall i \in [n] \\ x' \leftarrow \mathsf{Rec}(\{s_i\}_{i\in S}) \end{array} \right] = 1,$$

where the probability is taken over the random coins of Share.
If a negligible error probability is allowed, we simply say that the scheme is correct.

Privacy. The privacy property requires that any combination of up to t shares should leak no information about the secret x. More formally, we say that a secret sharing scheme is *private* if for all (unbounded) adversaries $\mathcal{A}$, for any set $\mathcal{I} \subseteq \{1, \ldots, n\}$, $|\mathcal{I}| \leq t$ and any two secrets x_0, x_1 (such that $|x_0| = |x_1|$),

$$\left| \Pr\left[\mathcal{A}(S) = 1 \,\middle|\, \begin{array}{c} \{s_i\}_{i\in[n]} = \mathsf{Share}(x_0); \\ S = \{s_i\}_{i\in\mathcal{I}} \end{array} \right] - \Pr\left[\mathcal{A}(S) = 1 \,\middle|\, \begin{array}{c} \{s_i\}_{i\in[n]} = \mathsf{Share}(x_1); \\ S = \{s_i\}_{i\in\mathcal{I}} \end{array} \right] \right| \leq \mu(\kappa)$$

for a negligible function μ in the bit-length κ of the size of $\mathbb{F}$.
Share Simulatability. Additionally, we require an efficient simulator for the generated shares. More formally, we say that a secret sharing scheme is *share simulatable* if there exists a PPT simulator $\mathsf{SimShare}$ such that for every PPT adversary $\mathcal{A}$, for any set $\mathcal{I} \subseteq \{1, \ldots, n\}$, $|\mathcal{I}| \leq t$ (and $\mathcal{H} = \{1, \ldots, n\} \backslash \mathcal{I}$), and any two secrets x_0, x_1, for $(s_0, \ldots, s_n) \leftarrow \mathsf{Share}(x_0)$, $(s'_1, \ldots, s'_n) \leftarrow \mathsf{Share}(x_1)$ and $\{s''_i\}_{i\in\mathcal{H}} \leftarrow \mathsf{SimShare}(\{s_i\}_{i\in\mathcal{I}}, x_0)$,

$$|\Pr[\mathcal{A}(\{s_i\}_{i\in\mathcal{I}}, \{s_i\}_{i\in\mathcal{H}}) = 1] - \Pr[\mathcal{A}(\{s_i\}_{i\in\mathcal{I}}, \{s''_i\}_{i\in\mathcal{H}}) = 1]| \leq \mu(\kappa)$$

for a negligible function μ in the bit-length κ of the size of $\mathbb{F}$.

Instantiation. In our constructions, we use Shamir's threshold secret sharing scheme [Sha79], and refer to its algorithms as ($\mathsf{SH.Share}, \mathsf{SH.Rec}, \mathsf{SH.Eval}, \mathsf{SH.SimShare}$). Shamir secret sharing satisfies the useful property we require in our construction – $\mathsf{SH.Eval}$ involves only linear operations (which is important since our construction executes $\mathsf{SH.Eval}$ on the threshold shares under the hood of a linearly homomorphic encryption scheme).

A.2 Linearly Homomorphic Encryption

A linearly homomorphic encryption scheme is a public key encryption scheme that allows any third party to take ciphertexts $\beta_1, \ldots, \beta_m$ decrypting to $x_1, \ldots, x_m$, as well as coefficients $c_1, \ldots, c_m$, and compute a ciphertext β that decrypts to $\sum_{j=1}^{m} c_j x_j$ (over a ring $\mathbb{F}$).

Syntax. A homomorphic (over ring $\mathbb{F}$) encryption scheme has the following algorithms:

$\mathsf{Gen}(1^\kappa) \to (pk, sk)$**:** An algorithm that, given the security parameter, generates a public-secret key pair (pk, sk).
$\mathsf{Enc}(pk, x; \rho) \to \beta$**:** An algorithm that, given the public key, a message $x \in \mathbb{F}$ and randomness ρ, outputs an encryption β of x.
$\mathsf{Dec}(sk, \beta) \to x$**:** An algorithm that, given the secret key and a ciphertext β, outputs a decryption x of β.
$\mathsf{Eval}(pk, (\beta_1, \ldots, \beta_m), f = (c_1, \ldots, c_m)) \to \beta$**:** A deterministic algorithm that, given the public key, ciphertexts $\beta_1, \ldots, \beta_m$ corresponding to messages $x_1, \ldots, x_m \in \mathbb{F}^m$ and a linear function f described as coefficients $c_1, \ldots, c_m$, outputs a ciphertext β that encrypts $f(x_1, \ldots, x_m) = \sum_{j=1}^{m} c_j x_j \in \mathbb{F}$.

Properties. We require the following properties of a homomorphic encryption scheme:

Perfect Correctness. The perfect correctness property requires that decryption of honestly produced ciphertexts must return the appropriate message. More formally, an encryption scheme is *perfectly correct* if for any message $x \in \mathbb{F}$,

$$\Pr\left[\mathsf{Dec}(sk, \beta) = x \;\middle|\; \begin{array}{l}(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa) \\ \beta \leftarrow \mathsf{Enc}(pk, x)\end{array}\right] = 1,$$

where the probability is taken over the random coins of Gen and Enc.
Further, in a homomorphic encryption scheme, decryption must remain correct even after homomorphic evaluation. More formally, we require that for any set of values $x_1, \ldots, x_m$, and any linear function f described as coefficients $c_1, \ldots, c_m$,

$$\Pr\left[x' = \sum_{j=1}^{m} c_j x_j \;\middle|\; \begin{array}{l}(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa) \\ \beta_j \leftarrow \mathsf{Enc}(pk, x_j)\ \forall j \in [m] \\ \beta \leftarrow \mathsf{Eval}(pk, (\beta_1, \ldots, \beta_m), f = (c_1, \ldots, c_m)) \\ x' \leftarrow \mathsf{Dec}(sk, \beta)\end{array}\right] = 1,$$

where the probability is taken over the random coins of Gen and Enc.
Semantic Security. The semantic security property requires that an adversary cannot distinguish which among the two messages (that the adversary chooses) is encrypted in a given ciphertext. More formally, for all PPT adversaries $\mathcal{A}$, for $(x_0, x_1) \leftarrow \mathcal{A}(1^\kappa)$, if $|x_0| = |x_1|$,

$$\Pr\left[\mathcal{A}(pk, \beta) = b \;\middle|\; \begin{array}{l}(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa); b \leftarrow \{0,1\} \\ \beta \leftarrow \mathsf{Enc}(pk, x_b)\end{array}\right] \leq \frac{1}{2} + \mu(\kappa),$$

where the probability is taken over the random coins of Gen and Enc.
Key Binding. This is a new property we introduce, which requires that the correspondence between a secret and public key be checkable. In particular, we require the existence of the following algorithm:

$\mathsf{KeyMatches}(pk, sk) \rightarrow$ `accept`/`reject`**:** Checks whether a given public key pk and secret key sk correspond to one another.

For any $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$, we require $\mathsf{KeyMatches}(pk, sk) =$ `accept`. Furthermore, for any message x, for any ciphertext $\beta \leftarrow \mathsf{Enc}(pk, x)$, it should hold that for all keys sk' such that $\mathsf{KeyMatches}(pk, sk') =$ `accept`, it holds that $\mathsf{Dec}(sk, \beta) = \mathsf{Dec}(sk', \beta)$. (To weaken the definition, we might consider *efficiently computable* keys sk' instead.) This allows parties to "prove correct decryption".
One might think that we get the ability to prove correct decryption for free from perfect correctness. However, perfect correctness only considers the honestly generated decryption key; key binding mmakes sure the adversary cannot get away with using a different key, which might convincingly decrypt to something incorrect.

Instantiation. We rely on the linearly homomorphic cryptosystem of Castagnos and Laguillaumie [CL15], which uses class groups. This scheme has the same structure as ElGamal encryption; a secret key consists of an element x, with g^x (for some generator g) as part of the public key. Notice that, in particular, this scheme gives is the key binding property for free; it is easy to check this discrete log relationship within the $\mathsf{KeyMatches}$ algorithm.

Notice also that if we didn't get key binding for free, we could add key binding to any encryption scheme by including in the public key a perfectly binding commitment to the secret key. The new key generation algorithm Gen would do the following (building on the key generation algorithm Gen' of the original encryption scheme):

$\mathsf{Gen}(1^\kappa) \rightarrow (pk, sk)$:
- $(pk', sk') \leftarrow \mathsf{Gen}'(1^\kappa)$
- Choose randomness ρ
- $\gamma \leftarrow \mathsf{Commit}(sk'; \rho)$
- Return $(pk = (pk', \gamma), sk = (sk', \rho))$

The $\mathsf{KeyMatches}$ algorithm would then simply return `accept` if $\gamma = \mathsf{Commit}(sk', \rho)$, and `reject` otherwise.

A.3 Multi-string Non-Interactive Zero-knowledge Proofs.

Multi-string NIZK proofs [GO07] is a generalization of NIZK in the common reference string (CRS) model. Instead of having one trusted authority to generate the reference string, in the multi-string model several authorities generate the reference strings. The properties of Multi-string NIZK proofs are defined similarly to those of NIZKs in the CRS model, except that the notions of completeness, soundness and zero-knowledge are required to hold only if the number of common reference strings that are honestly generated is above a certain threshold.

Syntax. A multi-string NIZK for an NP relation $\mathcal{R}_\mathcal{L}$ has the following algorithms:

$\mathsf{mNIZK.Gen}(1^\kappa) \rightarrow crs$: An algorithm to generate a common reference string. In the multi-string model comprising of n common reference strings, we let $\overline{crs} = (crs_1, \ldots, crs_n)$ denote the vector of the n common reference strings.
$\mathsf{P}(\overline{crs}, \phi, w) \rightarrow \pi$**:** An algorithm run by the prover that, given the vector of common reference strings $\overline{crs}$, statement ϕ and the witness w outputs the proof π that $(\phi, w) \in \mathcal{R}_\mathcal{L}$.
$\mathsf{V}(\overline{crs}, \phi, \pi) \rightarrow$ `accept/reject`**:** An algorithm that, given the vector of common reference strings $\overline{crs}$, statement ϕ and the proof π verifies whether π proves the existence of a witness w such that $(\phi, w) \in \mathcal{R}_\mathcal{L}$.

The rest of the algorithms are only necessary for proofs of security, and will not be used in the real world:

$\mathsf{S}_1(1^\kappa) \rightarrow (crs, \tau)$**:** A simulation algorithm that generates a simulated reference string and a simulation trapdoor.

$\mathsf{S}_2(\overline{crs}, \phi, \overline{\tau}) \rightarrow \pi$**:** A simulation algorithm that, given the vector of common reference strings $\overline{crs}$, statement ϕ and a vector $\overline{\tau}$ containing t_z (where t_z is a pre-defined threshold for the multi-string NIZK proof system) simulation trapdoors for common reference strings in $\overline{crs}$, outputs a simulated proof of the existence of a witness w such that $(\phi, w) \in \mathcal{R}_\mathcal{L}$.

$\mathsf{E}_1(1^\kappa) \rightarrow (crs, \xi)$**:** A simulation algorithm that generates a simulated reference string and an extraction trapdoor.

$\mathsf{E}_2(\overline{crs}, \phi, \pi, \overline{\xi}) \rightarrow w$**:** An extraction algorithm that, given the vector of common reference strings $\overline{crs}$, statement ϕ, a valid proof π and a vector $\overline{\xi}$ containing t_s (where t_s is a pre-defined threshold for the multi-string NIZK proof system) extraction trapdoors for common reference strings in $\overline{crs}$, outputs a witness w such that $(\phi, w) \in \mathcal{R}_\mathcal{L}$.

$\mathsf{SE}_1(1^\kappa) \rightarrow (crs, \tau, \xi)$**:** A simulation algorithm that outputs a simulated reference string, a simulation trapdoor and an extraction trapdoor such that (crs, τ) is distribted as the output of S_1, and (crs, ξ) is distributed as the output of E_1.

Properties. We require the following properties from a (t_c, t_s, t_z, n) multi-string NIZK proof system for an NP relation $\mathcal{R}_\mathcal{L}$ (as defined in the work [GO07]).

(t_c, t_s, t_z, n)**-Completeness.** Informally, this property requires that if at least t_c out of n common reference strings are honest, then the prover holding a witness for the statement should be able to create a convincing proof. More formally, for all non-uniform polynomial time adversaries $\mathcal{A}$,

$$\Pr\left[\mathsf{V}(\overline{crs}, \phi, \pi) = 1 \,\middle|\, \begin{array}{l}(\overline{crs}, \phi, w) \leftarrow \mathcal{A}^{\mathsf{mNIZK.Gen}}(1^\kappa)\\ \pi \leftarrow \mathsf{P}(\overline{crs}, \phi, w)\end{array}\right] \geq 1 - \mu(\kappa)$$

where $\mathsf{mNIZK.Gen}$ on query i output $crs_i \leftarrow \mathsf{mNIZK.Gen}(1^\kappa)$, at least t_c of the crs_i's generated by $\mathsf{mNIZK.Gen}$ are included and $\mathcal{A}$ outputs $(\phi, w) \in \mathcal{R}_\mathcal{L}$, and μ is a negligible function in the security parameter κ.

We use multi-string NIZKs with *perfect* (t_c, t_s, t_z, n)-completeness for all $0 \leq t_c \leq n$ in our protocol. This means that even if the adversary chooses all common reference strings itself, we are guaranteed to output an acceptable proof when $(\phi, w) \in \mathcal{R}_\mathcal{L}$.

(t_c, t_s, t_z, n)**-Soundness.** Informally, this property requires that if at least t_s out of n common random strings are honestly generated, then an adversary cannot forge the proof. The adversary gets to see possible choices of correctly generated common reference strings and can adaptively choose n of them. It may also include up to $n - t_s$ fake common reference strings it itself chooses. More formally, for all adversaries $\mathcal{A}$,

$$\Pr[\mathsf{V}(\overline{crs}, \phi, \pi) = 1 \text{ and } \phi \notin \mathcal{L} : (\overline{crs}, \phi, \pi) \leftarrow \mathcal{A}^{\mathsf{mNIZK.Gen}}(1^\kappa)] \leq \mu(\kappa)$$

where $\mathsf{mNIZK.Gen}$ is an oracle that on query i outputs $crs_i \leftarrow \mathsf{mNIZK.Gen}(1^\kappa)$, the adversary outputs $\overline{crs}$ such that at least t_s of the crs_i's generated by $\mathsf{mNIZK.Gen}$ are included, and μ is a negligible function in the security parameter κ.

(t_c, t_s, t_z, n)**-Zero Knowledge.** Informally, this property requires that if t_z common reference strings are correctly generated, then the adversary learns nothing from the proof. As is standard in the zero-knowledge literature, we say that this is the case when the proof can be simulated given only the statement ϕ.
The definition of zero-knowledge is split into the following two parts.

Reference String Indistinguishability. This property simply says that the adversary cannot distinguish real common reference strings from simulated reference strings. More formally, for all non-uniform polynomial time adversaries $\mathcal{A}$,

$$\left| \begin{array}{c} \Pr[\mathcal{A}(crs) = 1 | crs \leftarrow \mathsf{mNIZK.Gen}(1^\kappa)] \\ - \Pr[\mathcal{A}(crs) = 1 | (crs, \tau) \leftarrow \mathsf{S}_1(1^\kappa)] \end{array} \right| \leq \mu(\kappa)$$

for a negligible function μ in the security parameter κ.

(t_c, t_s, t_z, n)*-Simulation Indistinguishability.* This property strengthens the standard definition of zero-knowledge and requires that even with access to the simulation trapdoors, the adversary cannot distinguish real proofs from simulated ones on a set of simulated reference strings. More formally, for all non-uniform polynomial time adversaries $\mathcal{A}$,

$$\left| \begin{array}{c} \Pr[\mathcal{A}(\pi) = 1 | (\overline{crs}, \overline{\tau}, \phi, w) \leftarrow \mathcal{A}^{\mathsf{S}_1}(1^\kappa); \pi \leftarrow \mathsf{P}(\overline{crs}, \phi, w)] \\ - \Pr[\mathcal{A}(\pi) = 1 : (\overline{crs}, \overline{\tau}, \phi, w) \leftarrow \mathcal{A}^{\mathsf{S}_1}(1^\kappa); \pi \leftarrow \mathsf{S}_2(\overline{crs}, \overline{\tau}, \phi)] \end{array} \right| \leq \mu(\kappa),$$

where S_1 on query i outputs $(crs_i, \tau_i) \leftarrow \mathsf{S}_1(1^\kappa)$, the adversary outputs $(\phi, w) \in \mathcal{R}_\mathcal{L}$ and $\overline{crs}, \overline{\tau}$ such that at least t_z of the crs_i's generated by S_1 are included and $\overline{\tau}$ contains t_z simulation trapdoors τ_i corresponding to crs_i's that have been generated by the oracle S_1, and μ is a negligible function in the security parameter κ.

(t_c, t_s, t_z, n)**-Knowledge.** Informally, this property requires the existence of probabilistic polynomial time algorithms E_1 and E_2 that can extract a witness from a valid proof.
Like the definition of zero-knowledge, the definition is split into two parts.

Reference String Indistinguishability. For all non-uniform polynomial time adversaries $\mathcal{A}$,

$$\left| \begin{array}{c} \Pr[\mathcal{A}(crs) = 1 | crs \leftarrow \mathsf{mNIZK.Gen}(1^\kappa)] \\ - \Pr[\mathcal{A}(crs) = 1 | (crs, \xi) \leftarrow \mathsf{E}_1(1^\kappa)] \end{array} \right| \leq \mu(\kappa)$$

where μ is a negligible function in the security parameter κ.

Extractability. For all non-uniform polynomial time adversaries $\mathcal{A}$,

$$\Pr\left[\mathsf{V}(\overline{crs},\phi,\pi)=1,(\phi,w)\notin\mathcal{R}_{\mathcal{L}}\middle|\begin{matrix}(\overline{crs},\phi,\pi)\leftarrow\mathcal{A}^{\mathsf{E}_1}(1^\kappa)\\ w\leftarrow\mathsf{E}_2(\overline{crs},\phi,w,\overline{\xi})\end{matrix}\right]\leq\mu(\kappa)$$

where E_1 is an oracle that returns $(crs_i,\xi_i)\leftarrow\mathsf{E}_1(1^\kappa)$, $\overline{\xi}$ contains at least t_s ξ_i's corresponding to the crs_i's generated by E_1, and μ is a negligible function in the security parameter κ.

(t_c,t_s,t_z,n)**-Simulation Soundness.** Informally, this property requires that an adversary cannot prove any false statement even after seeing simulated proofs of arbitrary statements. More formally, for all non-uniform polynomial time adversaries $\mathcal{A}$,

$$\Pr[\mathsf{V}(\overline{crs},\phi,\pi)=1,(\overline{crs},\phi,\pi)\notin Q,\phi\notin\mathcal{L}|(\overline{crs},\phi,\pi)\leftarrow\mathcal{A}^{\mathsf{S}_1,\mathsf{S}_2}(1^\kappa)]\leq\mu(\kappa)$$

where S_1 on query i returns $(crs_i,\tau_i)\leftarrow\mathsf{S}_1(1^\kappa)$, S_2 on query $(\overline{crs}_j,\phi_j)$ returns $\pi_j\leftarrow\mathsf{S}_2(\overline{crs}_j,\overline{\tau}_j,\phi_j)$ with $\overline{\tau}_j$ having simulation trapdoors for the crs_i's generated by S_1, the adversary produces $\overline{crs}_j$ containing at least t_s crs_i's generated by S_1, Q is the list of statements and corresponding proofs $(\overline{crs}_j,\phi_j,\pi_j)$ in the queries to S_2, and μ is a negligible function in the security parameter κ.

(t_c,t_s,t_z,n)**-Simulation Extractability.** Informally, this property requires that even after seeing many simulated proofs, whenever the adversary makes a new proof, we should be able to extract a witness. More formally, a multi-string NIZK proof system is (t_c,t_s,t_z,n)-simulation extractable if it has (t_c,t_s,t_z,n)-knowledge, is a (t_c,t_s,t_z,n)-NIZK proof (i.e. completeness, soundness and zero-knowledge hold for the relevant thresholds), and for all non-uniform polynomial time adversaries $\mathcal{A}$,

$$\Pr\left[\begin{matrix}\mathsf{V}(\overline{crs},\phi,\pi)=1,\\(\overline{crs},\phi,\pi)\notin Q,\\(\phi,w)\notin\mathcal{R}_{\mathcal{L}}\end{matrix}\middle|\begin{matrix}(\overline{crs},\phi,\pi)\leftarrow\mathcal{A}^{\mathsf{SE}'_1,\mathsf{S}_2}(1^\kappa),\\ w\leftarrow\mathsf{E}_2(\overline{crs},\phi,\pi,\overline{\xi})\end{matrix}\right]\leq\mu(\kappa)$$

where SE'_1 on query i returns (crs_i,ξ_i) from $(crs_i,\tau_i,\xi_i)\leftarrow\mathsf{SE}_1(1^\kappa)$, S_2 on query $(\overline{crs}_j,\phi_j)$ returns $\pi_j\leftarrow\mathsf{S}_2(\overline{crs}_j,\overline{\tau}_j,\phi_j)$ (where $\overline{\tau}_j$ contains t_z τ_i's corresponding to crs_i's in $\overline{crs}_j$ generated by SE_1), Q is the list of statements and corresponding proofs $(\overline{crs}_j,\phi_j,\pi_j)$ made by S_2, $\overline{\xi}$ contains the first t_s ξ_i's generated by SE_1 corresponding to crs_i's in $\overline{crs}$, and μ is a negligible function in the security parameter κ.
The above property of simulation extractability implies simulation-soundness.

Instantiation. In our constructions, we use the $(0,t_s,t_z,n)$ multi-string NIZK with $t_s=t_z=\lceil(n+1)/2\rceil$ of [GO07], which relies on enhanced trapdoor permutations and satisfies the properties outlined above.