Constant-Round 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. We demonstrate three different approaches: the first two (which we call YaOSO and YOSO-GLS) use two and three rounds of communication, respectively. Our third approach (which we call YOSO-LHE) uses $O(d)$ rounds, where d is the multiplicative depth of the circuit being evaluated; however, it can be used to bootstrap any constant-round YOSO protocol that requires setup, by generating that setup within YOSO-LHE. Though YOSO-LHE requires more rounds than our first two approaches, it may be more practical, since the zero knowledge proofs it employs are more efficient to instantiate.

As a contribution of independent interest, we introduce a *verifiable state propagation* UC functionality, which allows parties to send private message which are verifiably derived in the "correct" way (according to the protocol in question) to anonymous receivers. This is a natural functionality to build YOSO primitives on top of.

[*]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).

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) [12,20,30] 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 [28,1], and discovering the true extent of the gender wage gap across many different institutions [24].

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.* [17] 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 and on MPC protocols using those channels.

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. RACCs are modeled as 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.* [7], 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.* [18], 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 the original YOSO paper of Gentry *et al.* [17], RACCs were modeled as an ideal functionality that allows private communication to anonymous receivers. However, this ideal functionality doesn't allow senders to prove that the messages they sent follow the protocol instructions, which is often necessary for achieving guaranteed output delivery. To model this, we instead introduce a *verifiable state propagation* (VSP) functionality $\mathcal{F}_{\mathsf{VSP}}$, which additionally requires the sender to input a witness proving that her messages follow the protocol. We describe $\mathcal{F}_{\mathsf{VSP}}$ informally in Section 1.3, and formally in Section 2.3. In this work, we do not focus on the problem of instantiating $\mathcal{F}_{\mathsf{VSP}}$; we leave this to future work. Intuitively, it can be built using RACCs together with zero knowledge proofs. We instead study how to build YOSO MPC assuming that an instantiation of $\mathcal{F}_{\mathsf{VSP}}$ is available.

YOSO MPC Recently, the first YOSO MPC protocols have appeared in the literature [17,13]. 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.* [13]) This construction relies on specially tailored sum-checks on top of the BGW construction [6]. 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.* [17]) This construction is based closely on the CDN protocol [14]. 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.* [17]) 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.

1.2 Our Contributions

In this paper, we make several contributions. First, we formalize the ideal functionality (realized using RACCs) which most YOSO MPC constructions can run on top of. We call this functionality the *verifiable state propagation* functionality, denoted $\mathcal{F}_{\mathsf{VSP}}$. This fills a gap in the original YOSO paper [17].

Next, we describe several YOSO MPC protocols in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model. (We assume that a realization of $\mathcal{F}_{\mathsf{VSP}}$ is available which requires only a single round of communication for each call to the functionality; this can be achieved with the use of RACCs and zero knowledge proofs.) 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 thrice-over.

Our first constant-round setup-free YOSO MPC protocol is *YaOSO*; it compiles an underlying two-round non-YOSO MPC protocol by garbling its second-message function. YaOSO itself only requires two rounds, which is optimal.

Our second YOSO MPC protocol is *YOSO-GLS*; it builds on the GLS protocol [21]. YOSO-GLS requires between three and five rounds, depending on the setup we assume is available; Five rounds are necessary if only RACCs are available, and three rounds are achievable if a URS (uniform random string) is additionally present. YOSO-GLS uses complex assumptions such as threshold FHE; however, unlike YaOSO, it does not rely on generic compilation, and so may in practice be more efficient.

Our third YOSO MPC protocol is *YOSO-LHE*; it is based on YOSO-CDN [17], but avoids the use of a secret shared decryption key, which is the setup

YOSO MPC scheme	Security Guarantee	Number of Rounds	Number of Speakers	Setup	Computational Building Blocks
Fluid MPC [13]	Security with Abort	$O(d)$	$O(dn)$	**none**	**none**
YOSO-CDN [17]	**Guaranteed Output Delivery**	$d+3$	$m+(d+1)n+2dh$	CRS, distribution of shares to first committee	NIZK, LHTE
YOSO-IT [17]	**Guaranteed Output Delivery**	$O(d)$	$\mathsf{poly}(m,d,n)$	**none**	**none**
YaOSO (this work)	**Guaranteed Output Delivery**	**2**	$\mathbf{m+n}$	**none**	YGC, 2-round MPC
YOSO-GLS (this work)	**Guaranteed Output Delivery**	3	$m+n+h$	URS	TFHE
YOSO-GLS (this work)	**Guaranteed Output Delivery**	5	$m+2n+2h$	**none**	TFHE
YOSO-LHE (this work)	**Guaranteed Output Delivery**	$d+3$	$m+(d+1)n+(2d+1)h$	**none**	LHE
Bootstrapping (this work)	**Guaranteed Output Dtelivery**	$O(1)$	$O(m+n)$	**none**	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**.

that YOSO-CDN relies on. YOSO-LHE takes $O(d)$ rounds, where d is the multiplicative depth of the circuit being computed. However, YOSO-LHE also leads to a YOSO MPC with a constant number of rounds; YOSO-LHE can be used execute the setup for a constant-round YOSO protocol that uses threshold fully homomorphic encryption (TFHE). We call this the *bootstrapping* protocol.

1.3 Verifiable State Propagation

YOSO protocols are analysed within the YOSO model, discussed in Section 2, which significantly simplifies analysis by allowing the protocol designer to only consider *abstract* committees of roles rather than the entire set of parties in the system, only some of whom might find themselves on committees. Previous protocols were designed assuming access to two functionalities, providing point-to-point and broadcast communication respectively (where the point-to-point functionality could be realized using RACCs).

In the case of computationally secure protocols, a gap remained: the YOSO-CDN protocol circumvented the need for verifiability of messages sent via the point-to-point functionality by assuming access to encryption keys for each role, and using only the broadcast functionality to send ciphertexts along with zero knowledge proofs of correctness. However, explicit access to keys is not quite consistent with the exclusive use of the two functionalities available.

In this work, we instead explicitly model the verification of messages within the sending mechanism. We do this by introducing the verifiable state propagation functionality $\mathcal{F}_{\mathsf{VSP}}$, which supports both point-to-point and broadcast messages, while providing a mechanism for proving these messages are correctly produced. In practice this provides the same possibilities when designing protocols as using zero knowledge proofs together with access to explicit encryption keys; however, it allows for much simpler protocol descriptions.

We envision multi-string NIZKs [22] to be a useful tool to realize the verification actions of the $\mathcal{F}_{\mathsf{VSP}}$. The protocols could consider a committee whose roles each locally compute a CRS for the multi-string NIZK scheme. If we pick our committee size 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. The members of this committee do not need to receive messages, allowing them to be self-selecting through cryptographic sortition, as proposed for the nominating committees of [7].

For efficiency, some YOSO protocols (including YOSO-LHE) employ homomorphic computations on ciphertexts, through the use of either fully or partially homomorphic encryption. $\mathcal{F}_{\mathsf{VSP}}$ allows the sender to prove that she correctly computed her messages as a function of the messages she has received (and possibly additional input), but not as a function of messages of which she is not the recipient. To additionally model such homomorphic computation on others' messages, we introduce the $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ functionality, described in Section 5.1. $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ allows the sender to additionally prove that her messages were correctly computed based on messages previously sent to a given recipient by others, *even though she might not know the contents of those messages.*

VSP and the big picture. Consider what must be done to deploy YOSO protocols: the roles need to be mapped (*compiled*) onto the set of real (or *natural*) parties, while maintaining meaningful security guarantees. For a YOSO protocol Π realising a functionality $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model, one might imagine the possibility of designing a compiler such that $\mathsf{Compile}(\Pi)$ UC realises an equivalent functionality $\mathcal{F}^{Natural}$ in the $\mathcal{F}_{\mathsf{VSP}}^{Natural}$-hybrid model. If it were then possible to adapt the role assignment approaches of [7] or [18] such that they UC realise $\mathcal{F}_{\mathsf{VSP}}^{Natural}$, this would provide a complete path for transforming a protocol designed in the YOSO model to a UC secure natural protocol. Black-box use of $\mathcal{F}_{\mathsf{VSP}}$ would then serve to disentangle the design of YOSO protocols from the construction of role assignment mechanisms and RACCs, allowing the later to focus on realising $\mathcal{F}_{\mathsf{VSP}}^{Natural}$ to be compatible with existing protocols.

Note that the environment classes considered for the compiled and uncompiled protocols are fundamentally different, as the YOSO environment is restricted to random corruptions within the set of roles, while the natural environment may allow static or adaptive corruptions across the entire set of parties.

1.4 YaOSO: Technical Overview

Our first protocol is a two-round YOSO MPC protocol with guaranteed output delivery which we call *YaOSO*. We present it in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model. The optimality of two rounds follows from the fact that any one-round YOSO protocol would be susceptible to a residual function attack [23] (as the adversary could recompute first-round messages on behalf of corrupt parties, while keeping the honest parties' messages fixed, to obtain multiple evaluations of the function).

YaOSO is a general compiler that transforms any two-round broadcast non-YOSO MPC protocol that achieves semi-malicious security [1] in the dishonest majority setting, to a two-round YOSO MPC protocol in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model that achieves malicious security with guaranteed output delivery in the honest majority setting. Since the former can be instantiated using the protocols of [16,8] with semi-malicious security in the plain model, this yields a round-optimal YOSO MPC protocol in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model with guaranteed output delivery.

Assumptions Our compiler is based on threshold secret sharing and adaptive garbled circuits (Appendix A.2), which can be built from one-way functions. The underlying protocol in our compiler can be instantiated using the protocols of Garg *et al.* [16] and Benhamouda *et al.* [8], which rely on 2-round semi-malicious oblivious transfer (OT). [2]

Recap of the Compiler of Ananth *et al.* [2] The main idea of our compiler is to adapt the compiler of Ananth *et al.* [2] to the YOSO setting. We begin with a high-level description of the compiler of Ananth *et al.* and refer to their paper for further details. The compiler of Ananth *et al.* transforms any two-round n-party MPC protocol with security against semi-malicious adversaries (say Π_{sm}) to a two-round n-party MPC protocol with guaranteed output delivery against semi-malicious fail-stop adversaries corrupting $t < n/2$ parties (say Π'). In the first round of Π', each party broadcasts the first-round message of the underlying protocol Π_{sm}, along with an adaptive garbled circuit whose code would be used to compute their second-round message in Π_{sm}. Such a garbled circuit has the party's input and randomness hard-coded and takes as input the first-round messages she receives. Each of the labels corresponding to the first-round messages are threshold-shared (with threshold t) among the set of parties. In the second-round, parties broadcast the relevant share of the label, based on the first-round messages that were broadcast. The threshold sharing ensures that even if a party aborts in the second round, the labels corresponding to the first-round messages can be reconstructed and her garbled circuit can be evaluated to obtain her second-round messages. This achieves guaranteed output delivery.

[1] where semi-malicious security refers to security against an adversary that follows the protocol honestly but can choose bad random coins for each round.

[2] The construction of Garg *et al.* [16] is based on a two-round OT in the plain model which is secure against semi-malicious receiver and semi-honest sender. We refer to Garg *et al.* [16] for further details.

Adapting the Compiler to the YOSO Setting In the compiler of Ananth *et al.*, the same participants are involved in the input, computation and output phases; different roles are required to carry out these actions in the YOSO setting. In the YOSO setting, we employ one committee for each round; the *input* committee, and the *computation* committee. First, the members of the input committee carry out the actions of the first-round of the compiler of Ananth *et al.* (as described above); however, instead of sending the secret shares to one another, they send them to the members of the computation committee. Next, we observe that this gives the computation committee all of the information it needs in order to enable the public reconstruction of the output; the actions of the participants in the second round of the compiler of Ananth *et al.* depend only on the first round *public* transcript and the threshold shares received in the first round. This transfer of threshold shares can be done via the $\mathcal{F}_{\mathsf{VSP}}$ functionality, which also upgrades the security to the malicious setting, as the actions of the input and computation committee roles can now be verified.

Lastly, we note that, unlike the output of the compiler of Ananth *et al.*, the output of our compiler should now be publicly computable (since we do not wish to involve a third committee). So, output computation should not depend on the secret state of the roles who computed the earlier messages. To address this, we assume that the output computation of Π_{sm} does not require any secrets and relies on the *public* transcript alone. We note that this can be assumed without loss of generality, as one can always consider the output computation executed by an additional participant of the MPC protocol Π_{sm} with a dummy input that uses a default random tape [3]. This completes the overview of our compiler, which is formally described in Section 3.

Round and Communication Complexity YaOSO uses two committees: an input committee of size m (where honest majority is not required) and a computation committee of size n (where the size n is chosen to be large enough to guarantee an honest majority within the committee). The protocol comprises just two rounds, where input committee roles speak first, followed by the computation committee roles. The communication complexity is the communication complexity of the underlying semi-malicious protocol that is being compiled, with an additional overhead of $O(|C|)$, where C is the circuit computing the next-message function for computing the second-round messages of the underlying protocol. (We do not explicitly consider the overhead incurred through the use of $\mathcal{F}_{\mathsf{VSP}}$.)

1.5 YOSO-GLS: Technical Overview

The second protocol we present is closely based on the three-round MPC protocol of Gordon *et al.* [21], which we will henceforth refer to as the GLS protocol. We call our adaptation *YOSO-GLS*. We present two variants of YOSO-GLS: the

[3]Since Π_{sm} is semi-maliciously secure, correctness of the output holds for any choice of random tape.

first requires three rounds and access to a uniform random string (URS), and the second requires two additional rounds instead sampling this string explicitly. We prove our YOSO-GLS protocol securely YOSO realises MPC with guaranteed output delivery in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model.

Assumptions The technical cornerstone of the GLS protocol is a threshold fully homomorphic encryption (TFHE) scheme based on the Learning with Error (LWE) assumption described in Definition 6. Our YOSO-GLS protocol relies similarly on this assumption.

Recap of the GLS Protocol To provide context for the necessary changes when adapting to the YOSO setting we start by providing a high level recap of the three round protocol of Gordon *et al.* and refer to their paper for further details.

> **Round 1:** In the first round, each party generates a key pair for the [19] FHE scheme, using a common matrix $\mathbf{B}$. Each party then broadcasts their generated public key. Note that $\mathbf{B}$ is assumed to be chosen uniformly at random and is available as a common reference string.
>
> **Round 2:** In the second round, parties distribute Shamir sharings of their secret keys along with a sharing of an additional error term. Parties then encrypt their input under their own public key, reusing the encryption randomness to produce additional hints, which allow transforming the ciphertext to an encryption under a common public key. These ciphertexts and hints are then broadcast.
>
> **Round 3:** In the third round, the ciphertexts are transformed to encryptions under the common public key for parties which appropriately distributed their key and error shares. The circuit may then be evaluated homomorphically on the ciphertexts. The resulting output is then partially decrypted by each party, exploiting the structure of the secret key sharings and linearity of decryption. The shares of error terms, distributed along with the secret keys in round one, are added to mask the partial decryptions. These partial decryptions are then broadcast.

Finally, parties may perform polynomial interpolation over the partial decryptions to reconstruct the final output.

Adapting the GLS Protocol to the YOSO Setting Moving to the YOSO model poses a series of concrete challenges, as roles may not maintain state and communicate across multiple rounds. The original GLS protocol requires storing the secret keys generated in the first round, so they may be shared in the second. Delaying sharing allowed avoiding the need for point-to-point communication in the first round. In their setting, access to broadcast would allow distributing public keys in the first round, enabling point-to-point commnication from the second round onwards. In our setting, the $\mathcal{F}_{\mathsf{VSP}}$ functionality allows private communication to future committees in all rounds, meaning the secret keys may already be secret shared and distributed in the first round. We call the committee performing this task the *key generation committee*, and the next committee — which broadcasts encrypted inputs — the *input committee*. The final committee will be the *computation committee*.

The separation between key generation and input committee roles presents a new problem: if an input is encrypted under any single public key, that input is leaked directly to the role which generated the key. Therefore, to avoid leaking its input, a role must instead transform the ciphertext towards a common public key prior to broadcasting it. This is possible due to the changes we have already made, by sharing keys one round earlier.

These changes allow us to move to the YOSO setting while maintaining the round complexity by requiring only three sequential committees, including the input committee. Modifying key generation has the added benefit of making it a local process, simplifying the presentation of the algorithm. Key generation in the GLS protocol required access to a uniform reference string, this is unchanged for our three round protocol. We provide a protocol realising the required URS sampling, through the use of two additional committees, for a combined five rounds in total.

Communication and Round Complexity The three round YOSO-GLS protocol uses one dishonest majority key generation committee of size h, an input committee of size m and a final honest majority computation committee of size n. The five round protocol, which avoids the need for a URS, requires one additional dishonest majority committee followed by an honest majority committee. We maintain the asymptotic message complexity of the original GLS protocol.

1.6 YOSO-LHE: Technical Overview

The third protocol we present is structurally similar to the YOSO-CDN protocol of Gentry *et al.* [17]. We call it *YOSO-LHE*. Like YOSO-CDN, YOSO-LHE requires $O(d)$ rounds of communication, where d is the multiplicative depth of the circuit being computed. However, unlike YOSO-CDN, YOSO-LHE does not require the trusted distribution of an initial set of key share. In order decrease the number of rounds without relying on such initial trusted distribution, YOSO-LHE can be used to generate the setup necessary for some constant-round YOSO MPC protocol (e.g. one based on threshold fully homomorphic encryption). We elaborate on this in Section 1.7.

Though YOSO-LHE requires more rounds than YaOSO or YOSO-GLS, we believe it may be more efficient in practice, because of the simpler message validity relations it employs; $\mathcal{F}_{\mathsf{VSP}}$ for those relations is much more practical to instantiate than for those required by our other protocols. We prove our YOSO-LHE protocol securely YOSO realises MPC with guaranteed output delivery in the $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$-hybrid model.

Assumptions YOSO-LHE uses the $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ functionality to allow participants to perform homomorphic operations on messages intended for others. YOSO-LHE does not rely on any additional assumptions outside of $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$; however, it's worth noting that $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ for the linear homomorphism we require can be based on the encryption scheme of Castagnos and Laguillaumie [11], which in turn is based on the DDH assumption in the class group setting.

Recap of the YOSO-CDN protocol Since our protocol is closely related to the YOSO-CDN protocol of Gentry *et al.* [17], we recap CDN — and YOSO-CDN— here.

CDN [14] 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 role 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.

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 roles will contain at least one honest role 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.

Adapting YOSO-CDN 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 use $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ to enable linear computations on messages to individual recipients. This eliminates the need for a trusted setup. In order to provide a secret to a committee instead of an individual recipient, the secret is shared, and the individual shares are encrypted.

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 role 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 are accomplished by leveraging $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$, and through the linear homomorphisms of the secret sharing scheme. 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 YOSO-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 role re-share its share to the new committee.

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 role, rather than an honest majority, is needed). YOSO-LHE uses one committee of size n for each layer of multiplication, as well as one additional committee of size n to decrypt the output. Each multiplication requires two committees of size h to generate a Beaver triple.

Including m roles who speak to provide inputs to the computation, this makes the total number of roles who speak throughout the protocol equal to $m + (d + 1)n + 2dh$.

It might look like this necessitates $3d+3$ rounds of communication, but many of these committees can speak at the same time. The two committees generating Beaver triples for the first multiplication must both speak before the first multiplication can happen, but the roles 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. Of course, the committee who decrypts the output must speak last, which leaves us with $d+3$ 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. 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 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.

1.7 Achieving Setup-Free Constant-Round YOSO MPC from YOSO-LHE

Gentry *et al.* [17] 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 YOSO MPC whose round complexity is independent of the circuit being computed. 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. (Note that the number of rounds may still depend on the security parameter depending on the setup performed.) 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.)

Lastly, we point that an alternate approach to designing constant-round YOSO protocols could be via randomized encoding i.e. to first consider the low-degree randomized encoding of the function to be computed and subsequently use a YOSO MPC to realize this. Since this approach typically leads to an efficiency blowup, we believe the bootstrapping approach outlined above to be more promising towards designing efficient constant-round YOSO MPC.

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 [17] 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.* [7]

and Gentry *et al.* [18]. (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 [10], 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 with environment $\mathcal{E}$ and adversary $\mathcal{A}$, the PPT environment $\mathcal{E}$ provides the input $x = (x_1, \ldots, x_m)$ 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 roles faithfully follow the instructions of the protocol using the input they were given. 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 $REAL_{\Pi,\mathcal{A},\mathcal{E}}(1^\kappa)$ 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 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 on the input given to them by $\mathcal{E}$. 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 $\mathcal{F}_f$ for computation of f with guaranteed output delivery.* Let $IDEAL_{\mathcal{F}_f,\mathcal{S},\mathcal{E}}(1^\kappa)$ denote the random variable (over the random coins used by $\mathcal{S}$) representing $\mathcal{E}$'s output in the ideal world.

Definition 1 (UC Security [10]). *Let $f : (\{0,1\}^*)^m \to \{0,1\}^*$ 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 $REAL_{\Pi,\mathcal{A},\mathcal{E}}(1^\kappa)$ and $IDEAL_{\mathcal{F}_f,\mathcal{S},\mathcal{E}}(1^\kappa)$ are indistinguishable for any large enough security parameter κ.*

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 role 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 roles.

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.

2.3 Verifiable State Propagation

The $\mathcal{F}_{\mathsf{VSP}}$ functionality maintains two maps for messages between roles, a map y for point-to-point messages, and a map z for broadcast messages. When roles have completed their work, they may input a single SEND message to the functionality containing all point-to-point messages as well as a broadcast message. The role also provides a witness and relation along this input, allowing the functionality to verify that the messages satisfy some requirement. The statements for the considered relations may be divided into three parts, ϕ_{send}, $\phi_{broadcast}$ and $\phi_{receive}$. The first two contain the point-to-point and broadcast messages respectively. The third and final part of the statement, which is specified by the functionality, contains all messages sent directly to the role, allowing roles to prove that their messages are well-formed with respect to secret messages they have received. After receiving a send command from an honest role the functionality outputs a SPOKE token, killing the role.

Roles may read messages input to the functionality in the rounds after they were sent.

Functionality $\mathcal{F}_{\mathsf{VSP}}$

This ideal functionality has the following behaviour:

– Define a map $\mathcal{R} : \mathsf{Role} \to \mathsf{Rel}_\perp$. *Specify the relations the messages of each role must satisfy.*
– Initially create point-to-point and broadcast maps:
$y : \mathbb{N} \times \mathsf{Role} \times \mathsf{Role} \to \mathsf{Msg}_\perp$ where $y(r, \mathsf{R}, \mathsf{R}') = \perp$ for all $r, \mathsf{R}, \mathsf{R}'$
$z : \mathbb{N} \times \mathsf{Role} \to \mathsf{Msg}_\perp$ where $z(r, \mathsf{R}) = \perp$ for all r, R.
– On input $(\textsc{Send}, \mathsf{S}, ((\mathsf{R}_1, x_1), \ldots, (\mathsf{R}_k, x_k)), x, w)$ in round r proceed as follows:
  - Let $\phi_{send} = ((\mathsf{R}_1, x_1), \ldots, (\mathsf{R}_k, x_k))$ and $\phi_{broadcast} = x$.
  - Collect all $y_k \neq \perp$ for $r' < r, \mathsf{R}' \in \mathsf{Role}$ where $y(r', \mathsf{R}', \mathsf{S}) = y_k$ to produce a vector $\phi_{receive} = ((\mathsf{R}'_1, y_1), \ldots, (\mathsf{R}'_m, y_m))$.
  - If $((\phi_{send} || \phi_{receive} || \phi_{broadcast}), w) \notin \mathcal{R}(\mathsf{S})$ ignore the input.
  - Else:
    * For $i \in [k]$ update $y(r, \mathsf{S}, \mathsf{R}_i) = x_i$. *Store point to point messages from the role.*
    * Update $z(r, \mathsf{S}) = x$. *Store the broadcast message from the role.*
    * Output $(\mathsf{S}, ((\mathsf{R}_1, x_1), \ldots, (\mathsf{R}_k, x_k)), x, w)$ to $\mathcal{S}$. *Leak messages and the witness to the simulator in a rushing fashion.*

  If S is honest give $\textsc{Spoke}$ to S.
– On input $(\textsc{Read}, \mathsf{R}, \mathsf{S}, r')$ in round r where $r' < r$ for $x = y(r', \mathsf{S}, \mathsf{R})$ output x to R.
– On input $(\textsc{Read}, \mathsf{S}, r')$ in round r where $r' < r$ output $x = z(r', \mathsf{S})$ to R.

We prove each of our protocols secure in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model, supplanting the broadcast and point-to-point functionalities of [17].

3 YaoOSO

As outlined in the overview in Section 1.4, we present a compiler that transforms any two-round broadcast non-YOSO MPC protocol that achieves semi-malicious security with abort in the dishonest majority setting to a two-round YOSO MPC protocol in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model that achieves malicious security with guaranteed output delivery in the honest majority setting. This compiler adapts the approach of the compiler in [2] with suitable modifications to make it compatible with the YOSO setting.

3.1 Tools

The compiler uses the following tools:

Tools.
– A two-round m-party non-YOSO broadcast protocol Π_{sm} achieving semi-malicious security with abort against dishonest majority (such as the protocols of [16,8]). Π_{sm} is represented by the set of algorithms $\{\texttt{frst-msg}_i$,

$\texttt{snd-msg}_i, \mathsf{out}\}$, where $\texttt{frst-msg}_i$ computes P_i's first-round broadcast message; $\texttt{snd-msg}_i$ computes P_i's second messages; and out computes the output.

The syntax of the algorithms is as follows:

- $\texttt{frst-msg}_i(x_i, \rho_i) \to \mathsf{msg}_i^1$ produces the first-round broadcast message of party P_i to all parties.
- $\texttt{snd-msg}_i(x_i, \rho_i, \mathsf{msg}_1^1, \ldots, \mathsf{msg}_m^1) \to \mathsf{msg}_i^2$ produces the second-round broadcast message of party P_i to all parties.
- $\mathsf{out}(\mathsf{msg}_1^1, \ldots, \mathsf{msg}_m^1, \mathsf{msg}_1^2, \ldots, \mathsf{msg}_m^2) \to y$ produces the public output. As mentioned previously, it is without loss of generality to assume that the output computation requires only the public transcript.

– An adaptive garbling scheme $(\texttt{garble}, \texttt{eval}, \texttt{simGC})$ (Appendix A.2).

– A Shamir secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$ (Appendix A.1).

Notation. Let $\mathtt{C}_{i,x_i,\rho_i}(\mathsf{msg}_1^1, \ldots, \mathsf{msg}_m^1)$ (with hard coded values (x_i, ρ_i)) denote the boolean circuit that computes $\texttt{snd-msg}_i$. For simplicity assume each first round message is ℓ bits long, so each circuit has $\mathsf{L} = m \cdot \ell$ input bits. Let g be the size of a garbled $\mathtt{C}_i$.

3.2 Protocol

We introduce a relation for each of the committees in the protocol, allowing $\mathcal{F}_{\mathsf{VSP}}$ to enforce correct behaviour.

Below, is the relation corresponding to an input committee role I_j,

$$\mathcal{R}_{\mathsf{Input},j} = \left\{ \begin{array}{l|l} \begin{array}{l} \phi_{send} = \big(E_i, \{s_{j,l,i}^{(0)}, s_{j,l,i}^{(1)}\}_{l \in [\mathsf{L}]}\big)_{i \in [n]} \\ \phi_{receive} = \bot \\ \phi_{broadcast} = \big(\mathsf{msg}_j^1, \mathtt{GC}_j\big) \\ w = \begin{array}{l} \big(x_j, \rho_j, \rho_{j,gc}, \\ \{\rho_{j,l}^{(b)}\}_{b \in \{0,1\}, l \in [\mathsf{L}]}\big) \end{array} \end{array} & \begin{array}{l} \mathsf{msg}_j^1 \leftarrow \texttt{frst-msg}_j(x_j, \rho_j) \\ (\mathtt{GC}_j, \{K_{j,l}^{(0)}, K_{j,l}^{(1)}\}_{l \in [\mathsf{L}]}) \leftarrow \texttt{garble}(1^\lambda, \mathtt{C}_{j,x_j,\rho_j}, \rho_{j,gc}) \\ \Big\{(s_{j,l,1}^{(b)}, \ldots, s_{j,l,n}^{(b)}) \leftarrow \mathsf{Share}(K_{j,l}^{(b)}, \rho_{j,l}^{(b)})\Big\}_{b \in \{0,1\}, l \in [\mathsf{L}]} \end{array} \end{array} \right\}.$$

Below, is the relation corresponding to a computation committee role E_i,

$$\mathcal{R}_{\mathsf{Computation},i} = \left\{ \begin{array}{l|l} \begin{array}{l} \phi_{send} = \bot \\ \phi_{receive} = \begin{array}{l} \{s_{j,l,i}^{(0)}, s_{j,l,i}^{(1)}\}_{j \in I, l \in [\mathsf{L}]}, \\ \mathsf{msg}_1^1, \ldots, \mathsf{msg}_m^1 \end{array} \\ \phi_{broadcast} = \{s_{j,l,i}^{(b_l)}\}_{j \in I, l \in [\mathsf{L}]} \\ w = \bot \end{array} & \mathsf{msg}_1^1 || \ldots || \mathsf{msg}_m^1 := b_1, \ldots, b_{\mathsf{L}} \end{array} \right\}.$$

We present the protocol Π_{YaOSO} in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model.

Protocol Π_{YaOSO} : Input

This step is run by the input committee I of size m. The jth input role I_j (with input x_j) does the following:

- Compute the first round message of Π_{sm} using input x_j and randomness ρ_j as $\mathsf{msg}_j^1 \leftarrow \texttt{frst-msg}_i(x_j, \rho_j)$
- Garble the circuit computing the second round next-message function of Π_{sm} as $(\texttt{GC}_j, \vec{K}_j) \leftarrow \texttt{garble}(1^\lambda, \texttt{C}_{j,x_j,\rho_j}, \rho_{j,gc})$, where $\vec{K}_j = \{K_{j,l}^{(0)}, K_{j,l}^{(1)}\}_{l \in [\mathsf{L}]}$ and $\rho_{j,gc}$ denotes the randomness used for garbling.
- Compute t-out-of-n threshold sharing of the labels as $(s_{j,l,1}^{(b)}, \ldots, s_{j,l,n}^{(b)}) \leftarrow \mathsf{Share}(K_{j,l}^{(b)}, \rho_{j,l}^{(b)})$ (for $l \in [\mathsf{L}]$ and $b \in \{0,1\}$), where $\rho_{j,l}^{(b)}$ denotes the randomness used.
- Send input $\big(\textsc{Send}, I_j, \big((E_1, \{s_{j,l,1}^{(0)}, s_{j,l,1}^{(1)}\}_{l\in[\mathsf{L}]}), \ldots, (E_n, \{s_{j,l,n}^{(0)}, s_{j,l,n}^{(1)}\}_{l\in[\mathsf{L}]})\big), \big(\mathsf{msg}_j^1, \texttt{GC}_j\big), \big(x_j, \rho_j, \rho_{j,gc}, \{\rho_{j,l}^{(b)}\}_{b\in\{0,1\}, l\in[\mathsf{L}]}\big)\big)$ to $\mathcal{F}_{\mathsf{VSP}}$.

Protocol Π_{YaOSO} : Computation

This step is run by the computation committee E of size n. The ith role E_i does the following:

- Collect the broadcast messages $\big(\mathsf{msg}_j^1, \texttt{GC}_j\big)$ of the input role I_j by giving input $(\textsc{Read}, I_j, 1)$ to $\mathcal{F}_{\mathsf{VSP}}$.
- Collect the point-to-point message $\{s_{j,l,i}^{(0)}, s_{j,l,i}^{(1)}\}_{l\in[\mathsf{L}]}$ sent by I_j by giving input $(\textsc{Read}, E_i, I_j, 1)$ to $\mathcal{F}_{\mathsf{VSP}}$. Let I' denote the subset of input roles I_j for whom the above reads resulted in non-$\perp$ output.
- For $\alpha \in \{(j-1)\ell+1, \ldots, j\ell\}$, let b_α denote the αth bit in msg_j^1 (where msg_j^1 is replaced by default first-round message for $j \notin I'$). In other words, set $b_1, \ldots, b_{\mathsf{L}} := \mathsf{msg}_1^1 || \ldots || \mathsf{msg}_m^1$.
- Send input $(\textsc{Send}, E_i, \perp, \{s_{j,l,i}^{(b_l)}\}_{j\in I, l\in\{\mathsf{L}\}}, \perp)$ to $\mathcal{F}_{\mathsf{VSP}}$. (Assume the set of shares to be simply $\perp$ for $j \notin I'$).

Protocol Π_{YaOSO} : Output

The output can be computed by any party as follows:

- Collect the broadcast messages $\big(\mathsf{msg}_j^1, \texttt{GC}_j\big)$ of each input role I_j by inputing $(\textsc{Read}, I_j, 1)$ to $\mathcal{F}_{\mathsf{VSP}}$. Let I' denote the subset of input roles I_j for whom the above read resulted in non-$\perp$ output.
- Collect the broadcast messages $\{s_{j,l,i}^{(b_l)}\}_{j\in I, l\in[\mathsf{L}]}$ of each computation committee role E_i by inputing $(\textsc{Read}, E_i, 2)$ Let E' denote the subset of committee roles E_i for whom the above read resulted in non-$\perp$ output.
- For $j \in I'$,
  - Reconstruct the appropriate input label $K_{j,l}$ for $l \in [\mathsf{L}]$ as $K_{j,l} \leftarrow \mathsf{Rec}(\{s_{j,l,i}^{(b_l)}\}_{i\in E'})$.
  - Evaluate $\texttt{GC}_j$ to obtain msg_j^2 as $\mathsf{msg}_j^2 \leftarrow \texttt{eval}(\texttt{GC}_j, K_{j,1}, \ldots, K_{j,\mathsf{L}})$.
- Compute the output as $y \leftarrow \texttt{out}(\mathsf{msg}_1^1, \ldots, \mathsf{msg}_m^1, \mathsf{msg}_1^2, \ldots, \mathsf{msg}_m^2)$, where msg_j^1 and msg_j^2 for input roles $j \notin I'$ are computed using default input and randomness.

Theorem 1. *The protocol Π_{YaOSO} realises the MPC functionality $\mathcal{F}_f$ with guaranteed output delivery in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model.*

The proof of Theorem 1 may be found in Appendix C.1

4 YOSO-GLS

We now present our first protocol as outlined in Section 1.5, starting with the TFHE scheme which makes up its core.

4.1 Threshold Fully Homomorphic Encryption

For the YOSO-GLS protocol we will use an adaptation of the Threshold Fully Homomorphic Encryption (TFHE) scheme described by Gordon *et al.* in [21]. To simplify the security proof of the final protocol, and potentially ease future use of TFHE, we extract three security properties of the scheme. This diverges from the approach of [21], where the authors analyse the scheme directly within the security proof of the protocol.

As described in our technical overview, moving to the YOSO model requires a series of modifications to the TFHE scheme, these changes are reflected in the syntax we present now.

Syntax.

$\mathsf{Setup}(1^\kappa, n, d; \rho) \to \mathsf{pp}$: A setup algorithm parameterized by the size of an honest majority committee n, producing public parameters pp, which are given as an implicit argument to all subsequent algorithms.

$\mathsf{KGen}(\rho_i) \to (pk_i, sk_i)$: Given public parameters pp and randomness ρ_i, the key generation algorithm produces a public key pk_i and a secret key sk_i split into shares, such that $sk_i = (sk_{i,1}, \ldots, sk_{i,n})$.

$\mathsf{Enc}(\{pk_i\}_{i\in\mathcal{K}}, x; \rho) \to C$: Given a set of public keys $\{pk_i\}_{i\in\mathcal{K}}$ and a message x, the encryption algorithm encrypts to a ciphertext C under randomness ρ.

$\mathsf{Eval}(f, C_1, \ldots, C_m) \to C$: Homomorphically evaluates function f on input ciphertexts $C_1, \ldots, C_m$ to produce C.

$\mathsf{PDec}(\{pk_i\}_{i\in\mathcal{K}}, csk_j, C) \to d_j$: For a ciphertext C, encrypted under the public keys $\{pk_i\}_{i\in\mathcal{K}}$, and computation secret key $csk_j = \{sk_{i,j}\}_{i\in\mathcal{K}}$ this algorithm produces a partial decryption d_j.

$\mathsf{Combine}(\{pk_i\}_{i\in\mathcal{K}}, C, \{d_i\}_{i\in R}) \to \mathsf{out}$: Given a set of partial decryptions $\{d_i\}_{i\in R}$ of size at least $t+1$, decrypts ciphertext C to plaintext out

To satisfy security we require one additional algorithm, for simulating partial decryptions.

$\mathsf{SimPDec}(C, \{pk_i\}_{i\in\mathcal{K}}, \{sk_i\}_{i\in\mathcal{I}_{\mathsf{KGen}}}, \{(csk_j, d_j)\}_{j\in\mathcal{I}_{\mathsf{Computation}}}, \mathsf{out}) \to \{d_j\}_{j\in[n]\setminus\mathcal{I}_{\mathsf{Computation}}}$: Given a ciphertext C under public keys $\{pk_i\}_{i\in\mathcal{K}}$, along with partial decryptions for corrupt computation roles, and all secrets known to the corrupted roles, the partial decryption simulation algorithm produces partial decryptions d_j for $j \in [n]\setminus\mathcal{I}_{\mathsf{Computation}}$ which are indistinguishable from honestly produced partial decryptions.

TFHE Security We extract three properties which we show our TFHE scheme adapted for the YOSO setting satisfies. We define correctness (Definition 2), semantic security (Definition 3, Figure 3), and partial decryption simulatability (Definition 4, Figure 4), where semantic security and partial decryption simulatability use a common set of oracles defined in Figure 2. The oracles are specifically tailored to represent the corruption powers of the adversary over different committees in the YOSO model. One peculiarity of this is that our oracles do not allow an honestly generated key to be corrupted or leaked subsequently, as roles erase all private state prior to sending any messages.

We will first define correctness.

Definition 2 (Correctness). *A* TFHE *scheme is perfectly correct, if for all positive integers* $n, h, d, t < n/2$*, all functions* f *(computed by a circuit of depth* d *only containing NAND gates and), all* $\rho, \{\rho_i^{\mathsf{KGen}}\}_{i\in[h]}, \{(x_i, \rho_i^{\mathsf{Enc}})\}_{i\in[m]}$*, and non-empty sets* $\mathcal{K} \subset [h], R \subset [n]$ *where* $|R| > t$*:*

$$f(x_1, \ldots, x_m) = \mathsf{Combine}(\{pk_i\}_{i\in\mathcal{K}}, C, \{d_i\}_{i\in R}).$$

Where the inputs to $\mathsf{Combine}$ *are produced as:*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, n, d; \rho)$
- $(pk_i, sk_i) \leftarrow \mathsf{KGen}(\rho_i^{\mathsf{KGen}})$ *for* $i \in \mathcal{K}$
- $C_i \leftarrow \mathsf{Enc}(\{pk_i\}_{i\in\mathcal{K}}, x_i; \rho_i^{\mathsf{Enc}})$ *for* $i \in [m]$
- $C \leftarrow \mathsf{Eval}(f, C_1, \ldots, C_m)$
- $d_j \leftarrow \mathsf{PDec}(\{pk_i\}_{i\in\mathcal{K}}, \{sk_{i,j}\}_{i\in\mathcal{K}}, C)$ *for* $j \in R$.

For the purposes of our remaining definitions, we must first capture the corruption powers of the adversary. We do this by formalising three oracles, with access to common state. These oracles are:

- $\mathcal{O}\mathsf{KGen}(i)$: An oracle that generates a new honest key pair and registers it in the system.
- $\mathcal{O}\mathsf{KReg}(i, \rho_i)$: An oracle that allows registering a corrupt key pair, requiring the randomness used for its generation.
- $\mathcal{O}\mathsf{Corr}(j)$: An oracle which leaks the computation key for csk_j.

These oracles track which keys have been chosen by and leaked to the adversary, allowing thresholds on corruptions to be checked in our security games. We do not allow the adversary to corrupt an honestly generated key after the fact, as this local state, such as the randomness used in generation, would be deleted in the YOSO setting.

Definition 3 (Semantic Security). *A* TFHE *scheme is semantically secure under chosen plaintext attack if, for all PPT adversaries* $\mathcal{A}$*,*

$$\mathsf{Adv}^{IND-CPA}_{\mathcal{A},n,t,\mathsf{TFHE}}(\kappa) = \Pr[\mathcal{A} \text{ wins } \mathsf{Game}^{IND-CPA}_{\mathcal{A},n,t,\mathsf{TFHE}}(\kappa)] - \frac{1}{2} \le negl(\kappa)$$

for a negligible function negl in the security parameter κ*. Where* $\mathsf{Game}^{IND-CPA}_{\mathcal{A},n,t,\mathsf{TFHE}}(\kappa)$ *is defined as described in Figure 3*

```
$\mathcal{O}\mathsf{KGen}(i)$
  1 : if $i \in \mathcal{H}_{\mathsf{KGen}} \cup \mathcal{I}_{\mathsf{KGen}} \vee i \notin [n]$ : return $\perp$
  2 : $(pk_i, sk_i = (sk_{i,1}, \ldots, sk_{i,n})) \leftarrow \mathsf{KGen}()$
  3 : $\mathsf{L}_{keys} := \mathsf{L}_{keys} \cup \{(i, pk_i, sk_i)\}$
  4 : $\mathcal{H}_{\mathsf{KGen}} := \mathcal{H}_{\mathsf{KGen}} \cup \{i\}$
  5 : return $pk_i$
$\mathcal{O}\mathsf{KReg}(i, \rho_i)$
  1 : if $i \in \mathcal{H}_{\mathsf{KGen}} \cup \mathcal{I}_{\mathsf{KGen}} \vee i \notin [n]$ : return $\perp$
  2 : $(pk_i, sk_i) \leftarrow \mathsf{KGen}(\rho_i)$
  3 : $\mathsf{L}_{keys} := \mathsf{L}_{keys} \cup \{(i, pk_i, sk_i)\}$
  4 : $\mathcal{I}_{\mathsf{KGen}} := \mathcal{I}_{\mathsf{KGen}} \cup \{i\}$
$\mathcal{O}\mathsf{Corr}(j)$
  1 : if $j \notin [n]$ : return $\perp$
  2 : $\mathcal{I}_{\mathsf{Computation}} := \mathcal{I}_{\mathsf{Computation}} \cup \{j\}$
  3 : $csk_j \leftarrow \{sk_{i,j} | \exists (i, pk_i, (sk_{i,1}, \ldots, sk_{i,n})) \in \mathsf{L}_{keys}\}$
  4 : return $csk_j$
```

Fig. 2: Oracles used in the security games for TFHE schemes

Definition 4 (Partial Decryption Simulatability). *A* TFHE *scheme has partial decryption simulatability if, for all PPT adversaries $\mathcal{A}$, for all n, d and functions f (of only NAND gates and depth less than d),*

$$\mathsf{Adv}^{ParDecSim}_{\mathcal{A},n,d,f,\mathsf{TFHE}}(\kappa) = \Pr[\mathcal{A} \text{ wins } \mathsf{Game}^{ParDecSim}_{\mathcal{A},n,d,f,\mathsf{TFHE}}(\kappa)] - \frac{1}{2} \leq negl(\kappa)$$

for a negligible function negl in the security parameter κ, where $\mathsf{Game}^{ParDecSim}_{\mathcal{A},n,d,f,\mathsf{TFHE}}(\kappa)$ *is defined as described in Figure 4*

Instantiating Threshold Fully Homomorphic Encryption We provide an instantiation of TFHE in Appendix A.4, proving its security in Appendix A.5. A discussion of the changes required to adapt the original scheme to the YOSO setting may be found in our technical overview (Section 1.5).

4.2 The YOSO-GLS Protocol

We present two variants of our YOSO-GLS protocol, achieving different round complexities, depending on the allowed setup.

- A five round protocol. Two sequential committees, realising a subprotocol for sampling public parameters for the TFHE scheme, described in Appendix B.1. Followed by three rounds for the TFHE scheme, consisting of a key generation, input and computation committee, described in this section.

```
$\mathsf{Game}^{IND-CPA}_{\mathcal{A},n,t,\mathsf{TFHE}}(\kappa)$
1 :  $\mathcal{H}_{\mathsf{KGen}} := \emptyset;\ \mathcal{I}_{\mathsf{KGen}} := \emptyset;\ \mathcal{I}_{\mathsf{Computation}} := \emptyset; \mathsf{L}_{keys} := \emptyset$
2 :  $\mathcal{O} \leftarrow \{\mathcal{O}\mathsf{KGen}, \mathcal{O}\mathsf{KReg}, \mathcal{O}\mathsf{Corr}\}$
3 :  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, n)$
4 :  $x_0, x_1 \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp})$
5 :  $\mathcal{K} \leftarrow \mathcal{H}_{\mathsf{KGen}} \cup \mathcal{I}_{\mathsf{KGen}}$
6 :  $b \xleftarrow{\$} \{0,1\}$
7 :  $C \leftarrow \mathsf{Enc}(\{pk_i\}_{i \in \mathcal{K}}, x_b)$
8 :  $b' \leftarrow \mathcal{A}^{\{\mathcal{O}\mathsf{Corr}\}}(C)$
9 :  if $|\mathcal{H}_{\mathsf{KGen}}| = 0 : \mathcal{A}$ loses
10 : if $|\mathcal{I}_{\mathsf{Computation}}| > t : \mathcal{A}$ loses
11 : if $|x_0| \neq |x_1| : \mathcal{A}$ loses
12 : if $b = b' : \mathcal{A}$ wins
13 : else $: \mathcal{A}$ loses
```

Fig. 3: The semantic security game for TFHE schemes. See Figure 2 for a definition of the oracles provided to the adversary.

– A three round protocol. The explicit sampling of public parameters is replaced by access to a uniform reference string, leaving only the three rounds needed for the TFHE scheme.

The functionality for sampling public parameters may be defined as $\mathcal{F}^{\mathsf{TFHE}}_{\mathsf{Setup}}$ (realised in Appendix B.1). For now we assume access to this ideal functionality and proceed to design our main protocol.

Functionality $\mathcal{F}^{\mathsf{TFHE}}_{\mathsf{Setup}}$

– Run $\mathsf{pp} \leftarrow \mathsf{TFHE.Setup}(1^\kappa, n, d)$ and output pp to $\mathcal{S}$.
– On input (READ, R) output pp to R.

Notation. For the purposes of the three-round protocol we will consider three committees:

$K_1, \ldots, K_h$ denotes the key generation committee K (of size h).
$I_1, \ldots, I_m$ denotes the input committee I (of size m).
$E_1, \ldots, E_n$ denotes the computing committee E (of size n).

We introduce a relation for each of the committees in the protocol, allowing $\mathcal{F}_{\mathsf{VSP}}$ to enforce correct behaviour.

```
$\mathsf{Game}^{ParDecSim}_{\mathcal{A},n,d,f,\mathsf{TFHE}}(\kappa)$

1 :  $\mathcal{H}_{\mathsf{KGen}} := \emptyset;\ \mathcal{I}_{\mathsf{KGen}} := \emptyset;\ \mathcal{I}_{\mathsf{Computation}} := \emptyset; \mathsf{L}_{keys} := \emptyset$
2 :  $\mathcal{O} \leftarrow \{\mathcal{O}\mathsf{KGen}, \mathcal{O}\mathsf{KReg}, \mathcal{O}\mathsf{Corr}\}$
3 :  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, n)$
4 :  $\{(x_k, \rho_k)\}_{k\in[m]} \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp})$
5 :  $\mathcal{K} \leftarrow \mathcal{H}_{\mathsf{KGen}} \cup \mathcal{I}_{\mathsf{KGen}}$
6 :  $\textbf{for } j \in [n]:$
7 :    $csk_j \leftarrow \{sk_{i,j} | \exists (i, pk_i, (sk_{i,1}, \ldots, sk_{i,n})) \in \mathsf{L}_{keys}\}$
8 :  $\textbf{for } k \in [m]:$
9 :    $C_k \leftarrow \mathsf{Enc}(\{pk_i\}_{i\in\mathcal{K}}, x_k; \rho_k)$
10 : $C \leftarrow \mathsf{Eval}(f, C_1, \ldots, C_m)$
11 : $\mathsf{out} \leftarrow f(x_1, \ldots, x_m)$
12 : $b \xleftarrow{\$} \{0, 1\}$
13 : $\textbf{if } b = 0:$
14 :   $\textbf{for } j \in [n] \setminus \mathcal{I}_{\mathsf{Computation}}:$
15 :     $d_j \leftarrow \mathsf{PDec}(\{pk_i\}_{i\in\mathcal{K}}, csk_j, C)$
16 : $\textbf{else}:$
17 :   $\textbf{for } j \in \mathcal{I}_{\mathsf{Computation}}:$
18 :     $d_j \leftarrow \mathsf{PDec}(\{pk_i\}_{i\in\mathcal{K}}, csk_j, C)$
19 :   $\{d_j\}_{j\in[n]\setminus\mathcal{I}_{\mathsf{Computation}}} \leftarrow \mathsf{SimPDec}(C, \{pk_i\}_{i\in\mathcal{K}}, \{sk_i\}_{i\in\mathcal{I}_{\mathsf{KGen}}},$
                 $\{(csk_j, d_j)\}_{j\in\mathcal{I}_{\mathsf{Computation}}}, \mathsf{out})$
20 : $b' \leftarrow \mathcal{A}(\{d_j\}_{j\in([n]\setminus\mathcal{I}_{\mathsf{Computation}})})$
21 : $\textbf{if } |\mathcal{H}_{\mathsf{KGen}}| = 0 : \mathcal{A}\ loses$
22 : $\textbf{if } |\mathcal{I}_{\mathsf{Computation}}| > t : \mathcal{A}\ loses$
23 : $\textbf{if } b = b' : \mathcal{A}\ wins$
24 : $\textbf{else}: \mathcal{A}\ loses$
```

Fig. 4: The partial decryption simulatability game for TFHE schemes. See Figure 2 for a definition of the oracles provided to the adversary.

Below, is the relation corresponding to a role in the key generation committee K_i,

$$\mathcal{R}_{\mathsf{KGen}} = \left\{ \begin{array}{l|l} \phi_{send} = \big((E_1, sk_{i,1}), \dots, (E_n, sk_{i,n})\big) & \\ \phi_{receive} = \perp & (pk_i, (sk_{i,1}, \dots, sk_{i,n})) \\ \phi_{broadcast} = \big(pk_i\big) & \leftarrow \mathsf{TFHE.KGen}(\mathsf{pp}, \rho_{K_i}) \\ w = \big(\rho_{K_i}\big) & \end{array} \right\}.$$

Below, is the relation corresponding to an input committee role I_i,

$$\mathcal{R}_{\mathsf{Enc}} = \left\{ \begin{array}{l|l} \phi_{send} = \perp & \\ \phi_{receive} = \perp & C_i \\ \phi_{broadcast} = \big(C_i\big) & \leftarrow \mathsf{TFHE.Enc}(\{pk_i\}_{i \in \mathcal{K}}, x_i; \rho_{I_i}) \\ w = \big(x_i, \rho_{I_i}\big) & \end{array} \right\}.$$

Lastly, the relation for a role in the computation committee E_i,

$$\mathcal{R}_{\mathsf{Eval}} = \left\{ \begin{array}{l|l} \phi_{send} = \perp & C \leftarrow \mathsf{TFHE.Eval}(f, C_1, \dots, C_m) \\ \phi_{receive} = \big((E_1, sk_{i,1}), \dots, (E_n, sk_{i,n})\big) & csk_i = \{sk_{j,i}\}_{j \in K} \\ \phi_{broadcast} = \big(d_i\big) & d_i \leftarrow \mathsf{PDec}(\{pk_k\}_{k \in \mathcal{K}}, csk_i, C) \\ w = \perp & \end{array} \right\}.$$

The $\mathcal{F}_{\mathsf{VSP}}$ functionality is parameterised by a map from roles to relations which their messages must satisfy. In this case, when defining our use of $\mathcal{F}_{\mathsf{VSP}}$, let $\mathcal{R}$ be the map such that $\mathcal{R}(K_i) = \mathcal{R}_{\mathsf{KGen}}$ for $i \in h$, $\mathcal{R}(I_j) = \mathcal{R}_{\mathsf{Enc}}$for $j \in m$, and $\mathcal{R}(E_k) = \mathcal{R}_{\mathsf{Eval}}$ for $k \in n$. Having defined our committees and necessary relations we may now present our protocol.

Protocol $\Pi_{YOSO-GLS}$

Setup: Any role R may read the public parameters pp after round two, by giving input (Read, R) to $\mathcal{F}^{\mathsf{TFHE}}_{\mathsf{Setup}}$.

KGen: *This step is run by the key generation committee K of size h.* Each member K_i ($i \in [h]$) does the following:
1. Runs the key generation algorithm $(pk_i, (sk_{i,1}, \dots, sk_{i,n})) \leftarrow \mathsf{TFHE.KGen}(\mathsf{pp}, \rho_{K_i})$.
2. Input $(\textsc{Send}, K_i, ((E_1, sk_{i,1}), \dots, (E_n, sk_{i,n})), pk_i, \rho_{K_i})$ to $\mathcal{F}_{\mathsf{VSP}}$.

When the key generation committee is finished all roles may define $\mathcal{K}$ such that $\{pk_i\}_{i \in \mathcal{K}}$ contains all keys where $pk_i \neq \perp$ is returned by $\mathcal{F}_{\mathsf{VSP}}$ on the input $(\textsc{Read}, K_i, 3)$.

Input: *This step is run by the input generation committee I of size m.* Each member I_i ($i \in [m]$) encrypts their input x_i under the TFHE keys to get $C_i \leftarrow \mathsf{TFHE.Enc}(\{pk_i\}_{i \in \mathcal{K}}, x_i; \rho_{I_i})$ and then inputs $(\textsc{Send}, I_i, \perp, C_i, (x_i, \rho_{I_i}))$ to $\mathcal{F}_{\mathsf{VSP}}$.

Computation: *This step is run by the computation committee E of size n.* Each member E_i ($i \in [n]$) does the following:
1. Collects all $C_j \neq \perp$ broadcasted by the committee I by giving input $(\textsc{Read}, I_j, 2)$ to $\mathcal{F}_{\mathsf{VSP}}$.
2. Evaluates the function f homomorphically on the ciphertexts $C \leftarrow \mathsf{TFHE.Eval}(f, C_1, \dots, C_m)$, replacing any missing C_j with default values.
3. Retrieves each key share by K, $sk_{j,i}$ by giving input $(\textsc{Read}, I_j, E_i, 2)$ to $\mathcal{F}_{\mathsf{VSP}}$ and defines $csk_i = \{sk_{j,i}\}_{j \in K}$.
4. Produces partial decryption $d_i \leftarrow \mathsf{PDec}(\{pk_k\}_{k \in \mathcal{K}}, csk_i, C)$.
5. Broadcasts d_i by inputing $(\textsc{Send}, E_i, \perp, d_i, \perp)$ to $\mathcal{F}_{\mathsf{VSP}}$.

Output: By inputing $(\textsc{Read}, E_i, 3)$ to $\mathcal{F}_{\mathsf{VSP}}$, any party may then take a set $\{d_i\}_{i \in R}$ of at least $t + 1$ partial decryptions and recover the final output $\mathsf{out} \leftarrow \mathsf{Combine}(\{pk_i\}_{i \in \mathcal{K}}, C, \{d_i\}_{i \in R})$.

Theorem 2. *The protocol $\Pi_{YOSO-GLS}$ YOSO realises the MPC functionality $\mathcal{F}_f$ with guaranteed output delivery in the $(\mathcal{F}_{\mathsf{VSP}}, \mathcal{F}_{\mathsf{Setup}}^{\mathsf{TFHE}})$-hybrid model.*

Here we provide a high level sketch of our proof strategy for the YOSO-GLS protocol. A complete proof of Theorem 2 may be found in Appendix C.2.

The $\mathcal{F}_{\mathsf{VSP}}$ functionality directly leaks inputs from the corrupt roles to the simulator, which may later input these to the MPC functionality to receive the result of the computation. The $\mathcal{F}_{\mathsf{VSP}}$ additionally forces the adversary to provide the randomness used for key generation and encryption to the simulator. For messages to be stored they must satisfy the relation specified for a given role. In our case, this ensures that keys, encryptions and decryptions are computed correctly, for some choice of randomness. Thus, correctness of our TFHE scheme implies that our protocol computes the correct output.

We may then proceed through two hybrids, reducing to the security properties of the TFHE scheme. First, partial decryptions for honest roles may be simulated, exploiting that the key shares for the corrupt roles have been leaked by $\mathcal{F}_{\mathsf{VSP}}$ and the output from the ideal MPC functionality. Indistinguishability follows from the partial decryption simulatability of the TFHE scheme, mapping corruptions in the protocol to use of the corresponding corruption oracles in the security game. In the second hybrid, bringing the simulator to the ideal world, encryptions of honest inputs may then be replaced by encryptions of zero. It is still possible to produce the same partial decryptions, simply simulating for the output received from the ideal functionality. Indistinguishability now follows from the semantic security of the TFHE scheme, again mapping corruptions in the protocol to oracle queries in the game.

5 YOSO-LHE

In this section, we describe our YOSO MPC protocol based on linearly homomorphic encryption (LHE). Instead of using LHE explicitly, we use the functionality $\mathcal{F}_{\mathsf{VSP}}^{hom}$, which models the use of homomorphic encryption for private communication to future roles.

5.1 $\mathcal{F}_{\mathsf{VSP}}^{hom}$: Homomorphic Verifiable State Propagation

In some cases the functionality of $\mathcal{F}_{\mathsf{VSP}}$ may be unnecessarily restrictive. For example, it is not unreasonable that the encryption scheme used to realise point-to-point communication have homomorphic properties. If this were the case then it might be possible for a role R' to apply a function on a message x, sent from a sender S to receiver R, without having to know what x is. We express this additional power by introducing an expanded variant of our verifiable state propagation functionality called $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$. The class of functions allowed by $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ may be restricted depending on the needs of the protocol and how the functionality is constructed, e.g. a linearly homomorphic encryption scheme allowing the application of any linear function.

Functionality $\mathcal{F}_{\mathsf{VSP}}^{hom}$

This ideal functionality has the following behaviour:

- Define a map $\mathcal{R} : \mathsf{Role} \to \mathsf{Rel}_{\perp}$. *Specify the relations the messages of each role must satisfy.*
- Initially create point-to-point and broadcast maps:
  $y : \mathbb{N} \times \mathsf{Role} \times \mathsf{Role} \to \mathsf{Msg}_{\perp}$ where $y(r, \mathsf{R}, \mathsf{R}') = \perp$ for all $r, \mathsf{R}, \mathsf{R}'$.
  In an abuse of notation we use $y(\mathsf{R})$ as a shorthand for the vector of all messages previously sent to *role*. Specifically, define $y(\mathsf{R})$ as the vector $((\mathsf{S}_1, y_1), \ldots, (\mathsf{S}_m, y_m))$ containing all pairs (S_j, y_j) such that $y_j = y(r', \mathsf{S}_j, \mathsf{R}) \neq \perp$ for some $r' < r$.
  $z : \mathbb{N} \times \mathsf{Role} \to \mathsf{Msg}_{\perp}$ where $z(r, \mathsf{R}) = \perp$ for all r, R.
- On input $(\textsc{Send}, \mathsf{S}, ((\mathsf{R}_1, x_1, f_1), \ldots, (\mathsf{R}_k, x_k, f_k)), x, w)$ in round r proceed as follows:
  - Let $\phi_{send} = ((\mathsf{R}_1, x_1, f_1), \ldots, (\mathsf{R}_k, x_k, f_k))$ and $\phi_{broadcast} = x$.
  - Collect all $y_k \neq \perp$ for $r' < r, \mathsf{R}' \in \mathsf{Role}$ where $y(r', \mathsf{R}', \mathsf{S}) = y_k$ to produce a vector $\phi_{receive} = ((\mathsf{R}'_1, y_1), \ldots, (\mathsf{R}'_m, y_m))$.
  - If $((\phi_{send} || \phi_{receive} || \phi_{broadcast}), w) \notin \mathcal{R}(\mathsf{S})$ ignore the input.
  - Else:
    * For $i \in [k]$ update $y(r, \mathsf{S}, \mathsf{R}_i) = (x_i, f_i(y(\mathsf{R}_i)))$. *Store point to point messages to each recipient role and apply the homomorphism on messages sent to each recipient role.*
    * Update $z(r, \mathsf{S}) = x$. *Store the broadcast message from the role.*
    * Output $(\mathsf{S}, ((\mathsf{R}_1, x_1, f_1), \ldots, (\mathsf{R}_k, x_k, f_k)), x, w)$ to $\mathcal{S}$. *Leak messages and the witness to the simulator in a rushing fashion.*

  If S is honest give $\textsc{Spoke}$ to S.
- On input $(\textsc{Read}, \mathsf{R}, \mathsf{S}, r')$ in round r where $r' < r$ for $(x, x_{hom}) = y(r', \mathsf{S}, \mathsf{R})$ output (x, x_{hom}) to R.
- On input $(\textsc{Read}, \mathsf{S}, r')$ in round r where $r' < r$ output $x = z(r', \mathsf{S})$ to R.

Realizing $\mathcal{F}_{\mathsf{VSP}}^{hom}$: Choosing Compatible Linearly Homomorphic Encryption and Secret Sharing. We wish to build our $YOSO-LHE$ scheme through the use of t-out-of-n secret sharings and a compatible linear homomorphism.

The natural choice of linearly homomorphic secret sharing is Shamir secret sharing [27]. 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 is strongly desirable that it not require a common reference string. Notably, well known encryption schemes such as Paillier [25] do not support distributed generation of keys, while alternate variants such as the cryptosystems due to Damgård and Jurik [15] and Bresson *et al.* [9] require a common reference string.

We instead propose the use of the linearly homomorphic cryptosystem of Castagnos and Laguillaumie [11], which has an ElGamal-like structure. The plaintext msg is encoded in an exponent (as f^{msg}) 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 msg using a generator f of a subgroup where the discrete logarithm problem is efficiently solvable. 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).[4]

5.2 Informal Overview of YOSO-LHE

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 $(x_1, \dots, x_n)$. She then uses $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ to send each share x_i to one of the members of C.

To perform linear operations, the members of a committee use the linear homomorphism of the Shamir secret sharing . 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 (of size h)— each of which only needs to have one honest role, 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 sends it via $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ to M, and independently shares and sends it via $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ to O. The value a is defined as the sum of successfully sent a_j's. We let $a_{j,k}$ denote the share sent by A_j to O_k.

Each member B_j of committee B then chooses a random value b_j, and proceeds similarly to the members of committee A. However, each role B_j also sends to each member O_k of O the value $b_j(a_{1,k} + \cdots + a_{h,k})$, using $\mathcal{F}_{\mathsf{VSP}}^{hom}$ to compute this value homomorphically without knowing the values $a_{i,k}$. To ensure that O_k cannot compute b_j by dividing the value it receives by $a_{1,k} + \cdots + a_{h,k}$ which it knows, B_j masks the value it sends with a freshly computed sharing of zero; so, what it actually sends is $b_j(a_{1,k} + \cdots + a_{h,k}) + 0_{j,k}$, where $0_{j,k}$ is the kth share of zero. As with a, b is defined as the sum of successfully sent b_j's; $c = ab$ is similarly defined as the sum of $b_j a$'s.

Next, to multiply two 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 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.)

The formal description of the YOSO-LHE may be found in Appendix B.2

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

Theorem 3. *The protocol $\Pi_{YOSO-LHE}$ realises the MPC functionality $\mathcal{F}_f$ with guaranteed output delivery in the $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$-hybrid model.*

The proof of Theorem 3 may be found in Appendix C.4.

References

1. SODA: Scalable oblivious data analytics. `https://soda-project.eu/` (2019)
2. Ananth, P., Choudhuri, A.R., Goel, A., Jain, A.: Round-optimal secure multiparty computation with honest majority. In: CRYPTO 2018, Part II (Aug 2018)
3. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: EUROCRYPT 2012 (Apr 2012)
4. Bellare, M., Hoang, V.T., Rogaway, P.: Adaptively secure garbling with applications to one-time programs and secure outsourcing. In: ASIACRYPT 2012 (Dec 2012)
5. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: ACM CCS 2012 (Oct 2012)
6. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC (May 1988)
7. Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., Reyzin, L.: Can a public blockchain keep a secret? In: TCC 2020, Part I (Nov 2020)
8. Benhamouda, F., Lin, H.: k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In: EUROCRYPT 2018, Part II (Apr / May 2018)
9. Bresson, E., Catalano, D., Pointcheval, D.: A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In: ASIACRYPT 2003 (Nov / Dec 2003)
10. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS (Oct 2001)
11. Castagnos, G., Laguillaumie, F.: Linearly homomorphic encryption from DDH. In: CT-RSA 2015 (Apr 2015)
12. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (abstract) (informal contribution). In: CRYPTO'87 (Aug 1988)
13. Choudhuri, A.R., Goel, A., Green, M., Jain, A., Kaptchuk, G.: Fluid MPC: Secure multiparty computation with dynamic participants. In: CRYPTO 2021, Part II (Aug 2021)
14. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: EUROCRYPT 2001 (May 2001)
15. Damgård, I., Jurik, M.: A length-flexible threshold cryptosystem with applications. In: ACISP 03 (Jul 2003)
16. Garg, S., Srinivasan, A.: Two-round multiparty secure computation from minimal assumptions. In: EUROCRYPT 2018, Part II (Apr / May 2018)
17. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: YOSO: You only speak once - secure MPC with stateless ephemeral roles. In: CRYPTO 2021, Part II (Aug 2021)
18. Gentry, C., Halevi, S., Magri, B., Nielsen, J.B., Yakoubov, S.: Random-index PIR and applications. In: TCC 2021, Part III (Nov 2021)

19. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: CRYPTO 2013, Part I (Aug 2013)
20. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: 19th ACM STOC (May 1987)
21. Gordon, S.D., Liu, F.H., Shi, E.: Constant-round MPC with fairness and guarantee of output delivery. In: CRYPTO 2015, Part II (Aug 2015)
22. Groth, J., Ostrovsky, R.: Cryptography in the multi-string model. In: CRYPTO 2007 (Aug 2007)
23. Halevi, S., Lindell, Y., Pinkas, B.: Secure computation on the web: Computing without simultaneous interaction. In: CRYPTO 2011 (Aug 2011)
24. Lapets, A., Jansen, F., Albab, K.D., Issa, R., Qin, L., Varia, M., Bestavros, A.: 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, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3209811.3212701, `https://doi.org/10.1145/3209811.3212701`
25. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT'99 (May 1999)
26. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: 37th ACM STOC (May 2005)
27. Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery (11) (Nov 1979)
28. Veeningen, M., Chatterjea, S., Anna Zsófia, H., Spindler, G., Boersma, E., van der Spek, P., van der Galiën, O., Gutteling, J., Kraaij, W., Veugen, T.: Enabling analytics on sensitive medical data with secure multi-party computation. Stud Health Technol Inform (2018)
29. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: 23rd FOCS (Nov 1982)
30. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS (Oct 1986)

Supplementary Material

A Tools

We recap the tools we need in our YOSO constructions.

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:

$\mathsf{Share}(x;\rho) \rightarrow (s_1,\ldots,s_n)$**:** An algorithm that, given a secret x, outputs a set of n shares in a finite field $\mathbb{F}$.

$\mathsf{Rec}(\{s_i\}_{i\in S\subseteq[n],|S|>t}) \rightarrow x$**:** An algorithm that, given a vector of at least $t+1$ shares, outputs the secret x.

$\mathsf{Eval}((s_1,\ldots,s_m),(c_1,\ldots,c_m)) \rightarrow s$**:** 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 $\mathbb{F}$.

$\mathsf{SimShare}(\{s_i\}_{i\in S,|S|\leq t}, x) \rightarrow \{s_i''\}_{i\in[n]\setminus S}$**:** 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 $|x_0| = |x_1|$),

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

for a negligible function $negl$ 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 $\{s''_i\}_{i \in \mathcal{H}} \leftarrow \mathsf{SimShare}(\{s_i\}_{i \in \mathcal{I}}, x_0)$,

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

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

Instantiation. In our constructions, we use Shamir's threshold secret sharing scheme [27], 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 Garbling Scheme

A garbling scheme, introduced by Yao [29] and formalized by Bellare *et al.* [5], enables a party to "encrypt" or "garble" a circuit in such a way that it can be evaluated on inputs — given tokens or "labels" corresponding to those inputs — without revealing what the inputs are.

Definition 5 (Garbling Scheme). *A projective garbling scheme is a tuple of efficient algorithms* $\mathtt{GC} = (\mathtt{garble}, \mathsf{Eval})$ *defined as follows.*

$\texttt{garble}(1^n, \texttt{C}) \to (\texttt{GC}, \mathbf{K})$**:** *The garbling algorithm* $\texttt{garble}$ *takes as input the security parameter* n *and a boolean circuit* $\texttt{C} : \{0,1\}^\ell \to \{0,1\}^m$*, and outputs a garbled circuit* $\texttt{GC}$ *and* ℓ *pairs of garbled labels* $\mathbf{K} = (K_1^0, K_1^1, \ldots, K_\ell^0, K_\ell^1)$*. For simplicity we assume that for every* $i \in [\ell]$ *and* $b \in \{0,1\}$ *it holds that* $K_\ell^b \in \{0,1\}^n$*.*

$\mathsf{Eval}(\texttt{GC}, K_1, \ldots, K_\ell) \to y$**:** *The evaluation algorithm* Eval *takes as input the garbled circuit* $\texttt{GC}$ *and* ℓ *garbled labels* $K_1, \ldots, K_\ell$*, and outputs a value* $y \in \{0,1\}^m$*.*

We require the following properties of a projective garbling scheme:

Perfect Correctness.. We say $\texttt{GC}$ *satisfies* perfect correctness *if for any boolean circuit* $\texttt{C} : \{0,1\}^\ell \to \{0,1\}^m$ *and* $x = (x_1, \ldots, x_\ell)$ *it holds that*

$$\Pr[\mathsf{Eval}(\texttt{GC}, \mathbf{K}[x]) = \texttt{C}(x)] = 1,$$

where $(\texttt{GC}, \mathbf{K}) \leftarrow \texttt{garble}(1^n, \texttt{C})$ *with* $\mathbf{K} = (K_1^0, K_1^1, \ldots, K_\ell^0, K_\ell^1)$*, and* $\mathbf{K}[x] = (K_1^{x_1}, \ldots, K_\ell^{x_\ell})$*.*

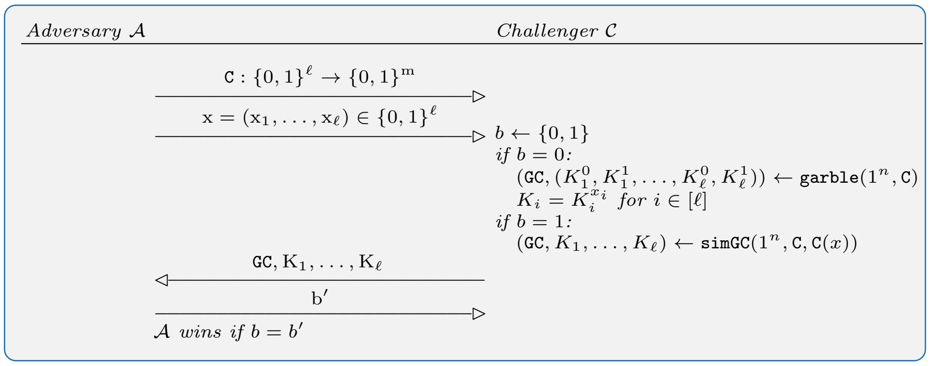
Next, we formally define the security notions we require for a garbling scheme. When garbled circuits are used in such a way that decoding information is used separately, obliviousness *requires that a garbled circuit together with a set of labels reveals nothing about the input the labels correspond to, and* privacy *requires that the additional knowledge of the decoding information reveals only the appropriate output. In our work, we do not consider decoding information separately (but rather, consider it to be included in the garbled circuit), so we do not need obliviousness.*

Privacy. Informally, privacy requires that a garbled circuit together with a set of labels reveal nothing about the input the labels correspond to (beyond the appropriate output).

More formally, we say that $\texttt{GC}$ *satisfies* privacy *if there exists a simulator* $\texttt{simGC}$ *such that for every PPT adversary* $\mathcal{A}$*, it holds that*

$$\Pr[\mathcal{A} \text{ wins}] \leq \frac{1}{2} + negl(n)$$

in the following experiment:

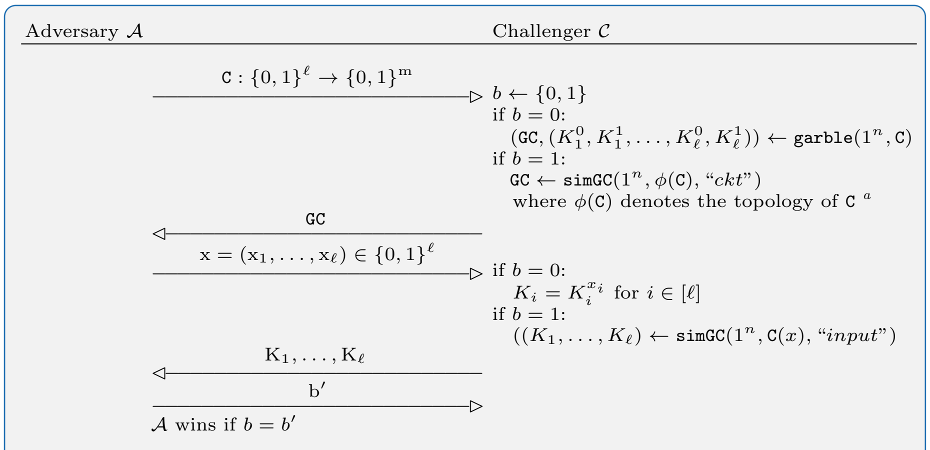

Adversary $\mathcal{A}$
Challenger $\mathcal{C}$
$\texttt{C} : \{0,1\}^\ell \to \{0,1\}^m$
$x = (x_1, \ldots, x_\ell) \in \{0,1\}^\ell$
$\texttt{GC}, K_1, \ldots, K_\ell$
b'
$\mathcal{A}$ wins if $b = b'$

```
$b \leftarrow \{0,1\}$
if $b = 0$:
    $(\texttt{GC}, (K_1^0, K_1^1, \ldots, K_\ell^0, K_\ell^1)) \leftarrow \texttt{garble}(1^n, \texttt{C})$
    $K_i = K_i^{x_i}$ for $i \in [\ell]$
if $b = 1$:
    $(\texttt{GC}, K_1, \ldots, K_\ell) \leftarrow \texttt{simGC}(1^n, \texttt{C}, \texttt{C}(x))$
```

Adaptive Privacy. Informally, this property requires that privacy is maintained against an adversary who first obtains the garbled circuit and then selects the input. More formally, we say that $\texttt{GC}$ satisfies *adaptive privacy* if there exists a simulator $\texttt{simGC}$ such that for every PPT adversary $\mathcal{A}$, it holds that

$$\Pr[\mathcal{A} \text{ wins}] \leq \frac{1}{2} + negl(n)$$

in the following experiment:



Adversary $\mathcal{A}$
Challenger $\mathcal{C}$
$\texttt{C} : \{0,1\}^{\ell} \to \{0,1\}^{m}$
$\texttt{GC}$
$x = (x_1, \ldots, x_\ell) \in \{0,1\}^{\ell}$
$K_1, \ldots, K_\ell$
b'
$\mathcal{A}$ wins if $b = b'$

```
$b \leftarrow \{0,1\}$
if $b = 0$:
  $(\texttt{GC}, (K_1^0, K_1^1, \ldots, K_\ell^0, K_\ell^1)) \leftarrow \texttt{garble}(1^n, \texttt{C})$
if $b = 1$:
  $\texttt{GC} \leftarrow \texttt{simGC}(1^n, \phi(\texttt{C}), \text{“}ckt\text{”})$
  where $\phi(\texttt{C})$ denotes the topology of $\texttt{C}$ [a]

if $b = 0$:
  $K_i = K_i^{x_i}$ for $i \in [\ell]$
if $b = 1$:
  $((K_1, \ldots, K_\ell) \leftarrow \texttt{simGC}(1^n, \texttt{C}(x), \text{“}input\text{”})$
```

[a] we assume that the topology of a circuit does not reveal hard coded values (as hard coded values are essentially fixed input labels for some wires)

Instantiation. For our constructions, adaptive garbled circuits can be obtained using one-time pads with Yao's garbled circuits (as shown by Bellare *et al.* [4]).

A.3 Vector operations

Before we may proceed to defining the scheme we must first recall a number of vector operations, presented in [19]. Let $\ell = \lfloor \log q \rfloor + 1$ be the length of the bit representation of an integer for some modulus q. We may then define the following procedures acting on vectors $a \in \mathbb{Z}_q^v$ and $a' \in \mathbb{Z}_q^{v \cdot \ell}$, where arithmetic is over $\mathbb{Z}_q$.

- $\mathsf{BitDecomp}(a) = (a_{1,0}, \ldots, a_{1,\ell-1}, \ldots, a_{v,0}, \ldots, a_{v,\ell-1})$ where $a_{i,j}$ is the jth bit of the ith element of a, such that $a_i = \sum_{j=0}^{\ell-1} 2^j a_{i,j}$.
- $\mathsf{BitDecomp}^{-1}(a') = (\sum_{j=0}^{\ell-1} 2^j a'_{1,j}, \ldots, \sum_{j=0}^{\ell-1} 2^j a'_{v,j})$ for $a' = (a_{1,0}, \ldots, a_{1,\ell-1}, \ldots, a_{v,0}, \ldots, a_{v,\ell-1})$, while this is most naturally defined when a' is a binary vector it remains well defined when this is not the case.
- $\mathsf{Flatten}(a') = \mathsf{BitDecomp}(\mathsf{BitDecomp}^{-1}(a'))$ for non-binary a' this procedure outputs a binary vector which preserves some of the structure of a'. This is a central feature in how the GSW scheme limits error growth, see [19] for detailed exposition.

We also define the above procedures on matrices, by simply applying the procedure row by row.

A.4 Threshold Fully Homomorphic Encryption construction

We will now describe our Threshold Fully Homomorphic Encryption construction $\mathsf{TFHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{PDec}, \mathsf{Combine}, \mathsf{SimPDec})$

Public Parameters. Following the approach of [21] we define the following public parameters, with respect to security parameter κ. The number of roles participating in the key generation is the size of a helper committee h, these roles each produce keys shares for a committee of n roles. We then define a bound $d = \mathsf{poly}(\kappa)$ for the maximal circuit depth, along with a modulus $q = \mathsf{poly}(d, \kappa)$. Finally, we introduce a lattice dimension $v = v(d, n)$ and error distribution $\chi = \chi(\kappa, d, n)$ chosen such that they provide κ bits of security for the $LWE_{v,q,\chi}$ problem (See Definition 6). We set $u = O((v+n)\log q)$ and let ℓ be the bitlength of our modulus $\lfloor \log q \rfloor + 1$. For our B_χ-bounded error distribution χ, we may define a positive integer bound $B_{\mathsf{smug}} \in \mathbb{Z}$ subject to the constraints:

$$\frac{((v+1)\ell+1)^d \cdot n \cdot B_\chi}{B_{\mathsf{smug}}} = negl(\kappa), \;\; n \cdot B_{\mathsf{smug}} < q/8.$$

The final public parameter produced is a uniformly random matrix $\mathbf{B} \in \mathbb{Z}_q^{u \times v}$, to be used when generating public keys. Note that $\mathbf{B}$ is the only public parameter which may not be locally and deterministically derived from know parameters. We then have $\mathsf{pp} = (v, u, q, \chi, B_\chi, B_{\mathsf{smug}}, \mathbf{B}) \leftarrow \mathsf{Setup}(1^\kappa, n)$.

Key Generation. We combine the two rounds of key generation of the [21] scheme into a single procedure. $\mathsf{KGen}(\mathbf{B})$ where role i proceeds as follows:

- Sample s_i uniformly in $\mathbb{Z}_q^v$, sample error term $\tilde{e}$ from χ^u, and compute $pk_i = \mathbf{B} \cdot s_i + \tilde{e}$. If any sample from χ has absolute value larger than B_χ, it should be replaced 0 if this is the case. (Note that as χ is B_χ-bounded this only happens with negligible probability)
- Sample an error term r_i uniformly from $[-B_{\mathsf{smug}}, B_{\mathsf{smug}}]$
- Shamir share s_i and r_i with a t-of-n threshold to produce $(s_{i,1}, \ldots, s_{i,n})$ and $(r_{i,1}, \ldots, r_{i,n})$. Note that as s_i is a vector each share $s_{i,j}$ actually consists of v pointwise shares.
- Output $(pk_i, ((s_{i,1}, r_{i,1}), \ldots, (s_{i,n}, r_{i,n})))$

Encryption. To permit a separate key generation committee we combine the encryption and ciphertext transformation procedures, this is possible as consolidating key generation into a single rounds allows roles giving input to know which key shares are available at time of encryption. $\mathsf{Enc}(\{pk_i\}_{i \in \mathcal{K}}, x)$ proceeds as follows:

- First compute public key $pk = \sum_{i\in\mathcal{K}} pk_i$
- Sample $\mathbf{R}$ uniformly from $\{0,1\}^{(v+1)\ell\times u}$
- Compute and output $C = \mathsf{Flatten}(x \cdot I_{(v+1)\ell} + \mathsf{BitDecomp}(\mathbf{R}\cdot pk||\mathbf{R}\cdot\mathbf{B}))$

Evaluation. All ciphertexts provided are encrypted under the same public key $pk = \sum_{i\in\mathcal{K}} pk_i$, this corresponds directly to an encryption in the [19] scheme under pk. Therefore, any circuit f, made up of only NAND gates, may be evaluated homomorphically on the ciphertexts. We refer the reader to [19] and [21] for a detailed explanation.

Partial Decryption. Let $\beta = \lfloor\log(q/2)\rfloor$, such that $2^\beta \in (q/4, q/2]$. The β-th row of C may then be parsed as $C_\beta = (C_{\beta,1}||C_{\beta,2})$ where $C_{\beta,1} \in \mathbb{Z}_q^\ell$ and $C_{\beta,2} \in \mathbb{Z}_q^{v\cdot\ell}$.

The partial decryption of our scheme proceeds exactly as the first round of decryption in [21]. Partial decryption PDec, of a ciphertext under keys $\{pk_i\}_{i\in\mathcal{K}}$, for a party j, which has received key and noise shares $\{(s_{i,j}, r_{i,j})\}_{i\in\mathcal{K}}$ may be done by:

- Summing all key shares $z_j = \sum_{i\in\mathcal{K}} s_{i,j}$
- Computing and outputting partial decryption $d_j = \langle\mathsf{BitDecomp}^{-1}(C_{\beta,2}), z_j\rangle + \sum_{i\in\mathcal{K}} r_{i,j}$

Final decryption. Given at least t partial decryptions of C under keys $\{pk_i\}_{i\in\mathcal{K}}$ any party reconstruct the final decryption. In $\mathsf{Combine}(\{pk_i\}_{i\in\mathcal{K}}, C, \{d_i\}_{i\in R})$ a party starts by choosing a set $R' \subset R$ of size $t+1$. They may then use Lagrange polynomials μ_k to reconstruct $w = \sum_{k\in R'} \mu_k(0)d_k$. This is then finally used to output the decryption:

$$\left\lfloor \frac{\mathsf{BitDecomp}^{-1}(C_{\beta,1}) - w}{2^\beta} \right\rceil$$

Partial Decryption Simulatability. Given the secret and randomness shares of corrupt roles the partial decryptions of the honest roles may be simulated. We extract the approach taken by the simulator described in [21] for generating third round messages, defining a seperate $\mathsf{SimPDec}$ algorithm, which proceeds as follows:

The algorithm is given ciphertext C with corresponding plaintext out, along with secret and randomness shares for $i \in \mathcal{I}_{\mathsf{KGen}}$, in the form of $sk_i = ((s_{i,1}, r_{i,1}), \ldots, (s_{i,n}, r_{i,n}))$. The algorithm additionally recieves $(d_j, csk_j = \{(s_{i,j}, r_{i,j})_{i\in\mathcal{K}}\})$ for $j \in \mathcal{I}_{\mathsf{Computation}}$.

1. Partial decryptions are interpolated to produce $w = \langle\mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{i\in\mathcal{K}} s_i\rangle$ such that
$$\mathsf{out} = \left\lfloor \frac{\mathsf{BitDecomp}^{-1}(C_{\beta,1}) - w}{2^\beta} \right\rceil.$$

To decrypt to a desired output contributions of honest partial decryptions must be constructed, such that they interpolate to give

$$W = \mathsf{BitDecomp}^{-1}(C_{\beta,1}) - \left\langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{i \in \mathcal{I}_{\mathsf{KGen}}} s_i \right\rangle - 2^{\beta} \cdot \mathsf{out},$$

where s_i is reconstructed for each corrupt party in $\mathcal{I}_{\mathsf{KGen}}$, from the secrets $(s_{i,1}, \ldots, s_{i,n})$.

2. We will define Shamir shares α_i such that the honest decryptions ensure that all partial decryptions reconstruct to W when combined. We start by defining $\alpha_0 = W$, and then pick indices $V \subset [n] \setminus \mathcal{I}_{\mathsf{Computation}}$, such that $|V| + |\mathcal{I}_{\mathsf{Computation}}| = t$, so that we may construct α_j for $j \in V \cup \mathcal{I}_{\mathsf{Computation}}$. This leaves α_j well defined for the remaining roles, allowing them to be interpolated.
   - For $j \in \mathcal{I}_{\mathsf{Computation}}$: $z'_j = \sum_{i \in \mathcal{H}_{\mathsf{KGen}}} s_{i,j}$
   - For $j \in V$: z'_j sampled uniformly randomly
   - For $j \in V \cup \mathcal{I}_{\mathsf{Computation}}$:

$$\alpha_j = \langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), z'_j \rangle$$

   - Reconstruct the remaining shares for $t \in [n] \setminus (V \cup \mathcal{I}_{\mathsf{Computation}})$: $\alpha_t = \sum_{j \in \{0\} \cup V \cup \mathcal{I}_{\mathsf{Computation}}} \mu_j(t) \alpha_j$
3. For honest roles $j \in [n] \setminus \mathcal{I}_{\mathsf{Computation}}$ define

$$d_i = \langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), z_i \rangle + \alpha_i + \sum_{j \in \mathcal{K}} r_{j,i}$$

where $z_i = \sum_{j \in \mathcal{I}_{\mathsf{KGen}}} s_{j,i}$. Finally, output $\{d_j\}_{j \in [n] \setminus \mathcal{I}_{\mathsf{Computation}}}$

A.5 Security of the TFHE construction

Prerequisites To prove security of our TFHE scheme we must first introduce the Learning with Errors assumption on which it is based.

Definition 6 (Learning with Error assumption [26]). *For integers $v = v(\kappa)$, $q = q(\kappa)$ and distribution $\chi = \chi(\kappa)$, we say the $LWE_{v,q,\chi}$ assumption holds if for any $u \in \mathsf{poly}(\kappa)$, the distribution $(\mathbf{B}, \mathbf{B} \cdot s + \tilde{e})$ is computationally indistinguishable from $(\mathbf{B}, u)$, where $\tilde{e} \leftarrow \chi^u$, and $(\mathbf{B}, s, u)$ are sampled uniformly as $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{u \times v}$, $s \xleftarrow{\$} \mathbb{Z}_q^v$ and $u \xleftarrow{\$} \mathbb{Z}_q^u$.*

We further recall a variant of the leftover hash lemma, and a useful result by Asharov *et al.* that shows adding large noise may hide the small noise terms from homomorphic evaluation.

Lemma 1 ([21], Implicit in [26]). *Let v, χ, q be parameters such that the $LWE_{v,q,\chi}$, and let n be some integer polynomial in κ. Then for $u = O((v + n)\log q)$, for any vectors $b_1, \ldots, b_{n-1} \in \mathbb{Z}_q^u$, the distribution of $(\mathbf{B}, b, \mathbf{R} \cdot (b || \mathbf{B}), \mathbf{R}(b_1 || \ldots || b_{n-1}))$ is computationally indistinguishable from $(\mathbf{B}, b, \mathbf{U}, \mathbf{R} \cdot (b_1 || \ldots || b_{n-1}))$, where $\mathbf{B}$ is uniform over $\mathbb{Z}_q^{u \times v}$, b is uniform over $\mathbb{Z}_q^u$, $\mathbf{U}$ is uniform over $\mathbb{Z}_q^{(v+1)\ell \times u}$, and $\ell = \lfloor \log q \rfloor + 1$.*

Lemma 2 ([3]). *Let $B_1 = B_1(\kappa)$, $B_2 = B_2(\kappa)$ be positive integers, and let e_1 be an integer such that $|e_1| < B_1$. Then for e_2 sampled uniformly in the interval $[-B_2, B_2]$, the distribution of e_2 is statistically close to that of $e_1 + e_2$, if $B_1/B_2 = negl(\kappa)$.*

Security proofs

Theorem 4. *The TFHE scheme defined in Section A.4 satisfies correctness Definition 2*

Proof. We follow the lines of the correctness proof of [21]. Each ciphertext $C_k \leftarrow \mathsf{Enc}(\{pk_i\}_{i\in\mathcal{K}}, x_k; \rho_k)$ corresponds to a GSW ciphertext under the public key $(\mathbf{B}, pk)$ where $pk = \sum_{i\in\mathcal{K}} pk_i$, i.e.

$$C_k = \mathsf{Flatten}(x_k \cdot I_{(v+1)\ell} + \mathsf{BitDecomp}(\mathbf{R} \cdot pk || \mathbf{R} \cdot \mathbf{B})).$$

By the analysis of [19] we know $C \leftarrow \mathsf{Eval}(f, C_1, \ldots, C_m)$ is an encryption of $\mathsf{out} = f(C_1, \ldots, C_m)$, with appropriately bounded error. Specifically, there exists some error $\tilde{e}$ satisfying $|\tilde{e}| < ((v+1)\ell + 1)^d \cdot n \cdot B_\chi$ such that

$$\mathsf{BitDecomp}^{-1}(C_{\beta,1}) - \left\langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{i\in\mathcal{K}} s_i \right\rangle = 2^\beta \cdot \mathsf{out} + \tilde{e}.$$

The norm of $\tilde{e}$ is strictly bounded as the key generation procedure replacing any error terms which are too large by zero, this prevents the adversary choosing randomness which causes samples from χ to exceed B_χ. Inspection of the partial decryptions reveals that interpolating from any set of at least $t+1$ partial decryptions $d_j = \langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{i\in\mathcal{K}} s_{i,j} \rangle + \sum_{i\in\mathcal{K}} r_{i,j}$ yeilds

$$w = \left\langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{i\in\mathcal{K}} s_i \right\rangle + \sum_{i\in\mathcal{K}} r_i$$

as r_i are sampled from $[-B_{\mathsf{smug}}, B_{\mathsf{smug}}]$ then $|\sum_{i\in\mathcal{K}} r_i| \leq n \cdot B_{\mathsf{smug}} < q/8$ and thus

$$\left\lfloor \frac{\mathsf{BitDecomp}^{-1}(C_{\beta,1}) - w}{2^\beta} \right\rceil = \mathsf{out}.$$

Here we implicitly rely on $\tilde{e}/B_{\mathsf{smug}} = negl(\kappa)$. □

Theorem 5. *The TFHE scheme defined in Section A.4 has semantic security following from Definition 3*

Proof. We prove the semantic security of our encryption scheme through a series of hybrids, which we subsequently prove are indistinguishable.

Real H_0: The challenger is run as described in $\mathsf{Game}^{IND-CPA}_{\mathcal{A},n,t,\mathsf{TFHE}}(\kappa)$

Hybrid H_1: Instead of running $\mathcal{O}\mathsf{KGen}(i)$ as prescribed pk_i is sampled as a uniform vector in $\mathbb{Z}_q^v$. For each $j \in \mathcal{I}_{\mathsf{Computation}}$ define $s_{i,j}$ and $r_{i,j}$ as uniformly random shares. For each query to $\mathcal{O}\mathsf{Corr}(j)$, fix any previously undefined $s_{i,j}$ and $r_{i,j}$ for $i \in \mathcal{H}_{\mathcal{K}}$ as random shares, prior to outputting them.

Hybrid H_2: Instead of encrypting the ciphertext set $C = \mathsf{BitDecomp}(U)$ where $U \xleftarrow{\$} \mathbb{Z}_q^{(v+1)\ell \times (v+1)}$.

In H_2 the ciphertext C has the same distribution independent of b, therefore no adversary may win the game with probability greater than 1/2. We will now prove our sequence of hybrids are indistinguishable to any adversary which wins the game:

$H_0 \approx H_1$ To ensure correctness KGen replaces any samples from χ which have norm larger than B_χ by zero, this only happens with negligible probability for each sample, and thus remains negligible when union bounding across the samples for each $\mathcal{O}\mathsf{KGen}$ query. Therefore, the error distribution with replacement is statistically indistinguishable from the distribution without replacement. The computational indistinguishability of each pk_i from uniform vectors then follows directly from the LWE assumption (Definition 6). For an adversary to win the game $|\mathcal{I}_{\mathsf{Computation}}| \leq t$ must hold, therefore for $i \in \mathcal{H}_{\mathsf{KGen}}$ an adversary will never see more than t shares $s_{i,j}$ of s_i. These shares are distribted indetically to the random shares produced in H_1.

$H_1 \approx H_2$ Consider the distribution ciphertext $C = \mathsf{Flatten}(C') = \mathsf{BitDecomp}(\mathsf{BitDecomp}^{-1}(C'))$. We may restrict our focus to the distribution of $\mathsf{BitDecomp}^{-1}(C')$ as this fully determines the distribution of the final ciphertext C. By the linearity of $\mathsf{BitDecomp}^{-1}$ we may now consider the distribution of

$$\mathsf{BitDecomp}^{-1}(x \cdot I_{(v+1)\ell}) + \mathsf{BitDecomp}^{-1}(\mathsf{BitDecomp}(\mathbf{R} \cdot pk || \mathbf{R} \cdot \mathbf{B}))$$

$$= \mathsf{BitDecomp}^{-1}(x \cdot I_{(v+1)\ell}) + (\mathbf{R} \cdot pk || \mathbf{R} \cdot \mathbf{B})$$

Consider the distribution of $\mathbf{R} \cdot pk || \mathbf{R} \cdot \mathbf{B}$, for $i \in \mathcal{H}_{\mathsf{KGen}}$ this is may be rewritten as $\mathbf{R} \cdot pk_i || \mathbf{R} \cdot \mathbf{B} + \sum_{j \in \mathcal{K} \setminus \{i\}} (\mathbf{R} \cdot pk_j || \mathbf{0})$, where $\mathbf{0}$ is the all zero matrix. By Lemma 1 the ensemble $(\mathbf{B}, pk_i, R \cdot (pk_i || \mathbf{B}), \{R \cdot pk_j\}_{j \in \mathcal{K} \setminus \{i\}})$ is indistinguishable from $(\mathbf{B}, pk_i, U, \{R \cdot pk_j\}_{j \in \mathcal{K} \setminus \{i\}})$, for any choice of pk_j for $j \in \mathsf{KGen} \setminus \{i\}$, where U is uniform over $\mathbb{Z}_q^{(v+1)\ell \times (v+1)}$. Thus we may replace the ciphertext by the indistinguishable $\mathsf{BitDecomp}(U)$.

□

Theorem 6. *The TFHE scheme defined in Section A.4 has partial decryption simulatability Definition 4.*

Proof. We will prove security in the partial decryption simulatability game by showing the statistical distance between the real partial decryptions and simulated partial decryptions in negligible.

When the adversary is given the partial decryptions on behalf of the honest roles it no longer has access to the corruption oracles. Thus we may consider fixed sets $\mathcal{H}_{\mathsf{KGen}}, \mathcal{I}_{\mathsf{KGen}}$ from key generation as well as $\mathcal{I}_{\mathsf{Computation}}$. Let $\mathcal{K} = \mathcal{H}_{\mathsf{KGen}} \cup \mathcal{I}_{\mathsf{KGen}}$.

Consider the statistical distance $\Delta(X, Y)$ between distributions X, Y defined as:

$$X = \left\{ \left\langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{j \in \mathcal{K}} s_{j,i} \right\rangle + \sum_{j \in \mathcal{H}_{\mathsf{KGen}}} r_{j,i} \;\middle|\; i \in [n] \setminus \mathcal{I}_{\mathsf{Computation}} \right\}$$

$$Y = \left\{ \left\langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{j \in \mathcal{I}_{\mathsf{KGen}}} s_{j,i} \right\rangle + \alpha_i + \sum_{j \in \mathcal{H}_{\mathsf{KGen}}} r_{j,i} \;\middle|\; i \in [n] \setminus \mathcal{I}_{\mathsf{Computation}} \right\}$$

Observe that X is the distribution of partial decryptions produced by the PDec algorithm, while Y is the distribution of produced by the $\mathsf{SimPDec}$ algorithm, excluding the contributions due to shares of r_j for $j \in \mathcal{I}_{\mathsf{KGen}}$. It is clear that $\mathsf{Adv}^{ParDecSim}_{\mathcal{A},n,d,f,\mathsf{TFHE}}(\kappa) \leq \Delta(X, Y)$. Due to the linearity of the inner product we may simplify this further and instead consider X' and Y' where $\Delta(X, Y) = \Delta(X', Y')$ for

$$X' = \left\{ \left\langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{j \in \mathcal{H}_{\mathsf{KGen}}} s_{j,i} \right\rangle + \sum_{j \in \mathcal{H}_{\mathsf{KGen}}} r_{j,i} \;\middle|\; i \in [n] \setminus \mathcal{I}_{\mathsf{Computation}} \right\},$$

$$Y' = \left\{ \alpha_i + \sum_{j \in \mathcal{H}_{\mathsf{KGen}}} r_{j,i} \;\middle|\; i \in [n] \setminus \mathcal{I}_{\mathsf{Computation}} \right\}.$$

We may now observe that X' is distribted as random secret shares of

$$\left\langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{j \in \mathcal{H}_{\mathsf{KGen}}} s_j \right\rangle + \sum_{j \in \mathcal{H}_{\mathsf{KGen}}} r_j$$

and Y' is distributed as random secret shares of

$$W + \sum_{j \in \mathcal{H}_{\mathsf{KGen}}} r_j.$$

Following the argumentation of the correctness proof (Theorem 4) we know

$$\mathsf{BitDecomp}^{-1}(C_{\beta,1}) - \left\langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{j \in \mathcal{K}} s_j \right\rangle = 2^{\beta} \cdot \mathsf{out} + \tilde{e}$$

for $|\tilde{e}| < ((v+1)\ell + 1)^d \cdot n \cdot B_{\chi}$.

Therefore,

$$\left\langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{j \in \mathcal{H}_{\mathsf{KGen}}} s_j \right\rangle$$
$$= \mathsf{BitDecomp}^{-1}(C_{\beta,1}) - \left\langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{j \in \mathcal{I}_{\mathsf{KGen}}} s_j \right\rangle - 2^{\beta} \cdot \mathsf{out} + \tilde{e}$$
$$= W + \tilde{e}.$$

(Recall $W = \mathsf{BitDecomp}^{-1}(C_{\beta,1}) - \langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{i \in \mathcal{I}_{\mathsf{KGen}}} s_i \rangle - 2^{\beta} \cdot \mathsf{out}$)

As $|\mathcal{H}_{\mathsf{KGen}}| > 0$ we know by Lemma 2 that $\Delta(W + \tilde{e} + \sum_{j \in \mathcal{H}_{\mathsf{KGen}}} r_j, W + \sum_{j \in \mathcal{H}_{\mathsf{KGen}}} r_j)$ is negligible in the security parameter, concluding our proof. □

B YOSO constructions

B.1 Generating Public Parameters for YOSO-GLS

The public parameters of the TFHE scheme (Appendix A.4) include a uniformly sampled matrix $\mathbf{B}$, in this section we will demonstrate how to explicitly sample such a matrix in a two round YOSO protocol. All other public parameters may be derived deterministically. In the first committee, which may have a dishonest majority, roles may sample random matrices, producing index-wise Shamir sharings to send to the subsequent committee. The second committee which must have an honest majority, then adds shares they have received and broadcasts the result. Intuitively, this prevents a rushing adversary from biasing $\mathbf{B}$, as it would have to corrupt a majority of the second committee to know the contribution of the honest roles.

Notation For the purposes of our protocol we define two committees:

$S_1, \ldots, S_h$ denotes the sampling committee S (of size h).
$C, \ldots, C_n$ denotes the combining committee C (of size n).

To ensure correct behaviour of the roles in these committees we define the relations:

$$\mathcal{R}_{\mathsf{Share}} = \left\{ \begin{array}{l|l} \phi_{send} = \big(\mathbf{B}_{j,1}, \ldots, \mathbf{B}_{j,n}\big) & \\ \phi_{receive} = \bot & \mathbf{B}_{j,1}, \ldots, \mathbf{B}_{j,n} \\ \phi_{broadcast} = \bot & \leftarrow \mathsf{Share}(\mathbf{B}_j; \rho_{S_j}) \\ w = \big(\mathbf{B}_j, \rho_j\big) & \end{array} \right\},$$

$$\mathcal{R}_{\mathsf{Combine}} = \left\{ \begin{array}{l|l} \phi_{send} = \bot & \\ \phi_{receive} = \big(\mathbf{B}_{1,i}, \ldots, \mathbf{B}_{h,i}\big) & \mathbf{B}_i = \sum_{j=1}^{h} \mathbf{B}_{j,i} \\ \phi_{broadcast} = \mathbf{B}_i & \text{where } \mathbf{B}_{j,i} = \bot \text{ is replaced by } \mathbf{0} \\ w = \bot & \end{array} \right\}.$$

We now define our protocol Π_{Setup}.

Protocol Π_{Setup}

This subprotocol is run by two sequential committees S and C, of sizes h and n respectively.

Sample: Each member S_j ($j \in [h]$) of the committee S does the following:

1. Uniformly sample a matrix $\mathbf{B}_j \overset{\$}{\leftarrow} \mathbb{Z}_q^{u \times v}$
2. Computes a point-wise Shamir sharing of $\mathbf{B}_j$
   - $\mathbf{B}_{j,1}, \ldots, \mathbf{B}_{j,n} \leftarrow \mathsf{Share}(\mathbf{B}_j; \rho_{S_j})$
3. Input $(\textsc{Send}, S_j, ((C_1, \mathbf{B}_{j,1}), \ldots, (C_n, \mathbf{B}_{j,n})), \perp, \rho_{S_j})$ to $\mathcal{F}_{\mathsf{VSP}}$

Combine: Each member C_i ($i \in [n]$) of the committee C does the following:

1. Inputs $(\textsc{Read}, S_j, C_i, 1)$ for each $j \in [h]$ to get $(\mathbf{B}_{1,i}, \ldots, \mathbf{B}_{h,i})$, replacing any $\mathbf{B}_{j,i} = \perp$ with $\mathbf{0}$.
2. Defines $\mathbf{B}_i = \sum_{j \in [h]} \mathbf{B}_{j,i}$
3. Input $(\textsc{Send}, C_i, \perp, \mathbf{B}_i, \perp)$ to $\mathcal{F}_{\mathsf{VSP}}$

Output: The matrix $\mathbf{B}$ may then be publicly computed, where each index is defined as $\mathbf{B}^{\alpha,\beta} \leftarrow \mathsf{Rec}(\{\mathbf{B}_i^{\alpha,\beta}\}_{i \in R})$ where $R \subset [n]$ of size at least $t+1$, and $\mathbf{B}_i \neq \perp$ for $i \in R$. Each $\mathbf{B}_i$ may be read from $\mathcal{F}_{\mathsf{VSP}}$ by giving input $(\textsc{Read}, C_i, 2)$. All remaining public parameters may be locally derived given only κ, n and d. The roles may then output $\mathsf{pp} = (v, u, q, \chi, B_\chi, B_{\mathsf{smug}}, \mathbf{B})$.

Theorem 7. *The protocol Π_{Setup} YOSO-realises the functionality $\mathcal{F}_{\mathsf{Setup}}^{\mathsf{TFHE}}$ in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model.*

The proof of Theorem 7 may be found in Appendix C.3.

B.2 Formal Description of YOSO-LHE

We describe the formal protocol in the $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$-hybrid below, which uses the tool of Shamir secret sharing scheme ($\mathsf{Share}, \mathsf{Rec}$), described in Appendix A.1. Below, we describe our notation for the relevant roles.

- $C_{l,1}, \ldots, C_{l,n}$ denotes the roles of a generic committee C_l (of size n).
- $A_{l,1}, \ldots, A_{l,h}$ and $B_{l,1}, \ldots, B_{l,h}$ 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.

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.

Before describing the protocol, we introduce some relations for use in $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$. Below, is the relation corresponding to an input committee role.

$$\mathcal{R}_{\mathsf{Share}} = \left\{ \begin{array}{l} \phi_{send} = \big((C_1, x_1, \perp), \ldots, (C_n, x_n, \perp)\big) \\ \phi_{receive} = \perp \\ \phi_{broadcast} = \perp \\ w = \big(x, \rho\big) \end{array} \middle| (x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x; \rho) \right\},$$

Below, is the relation corresponding to roles that decrypt i.e. open a value which has been computed as a public linear function (denoted as f_ℓ) of a subset of the values that this role has received,

$$\mathcal{R}_{\mathsf{Dec}} = \left\{ \begin{array}{l} \phi_{send} = \perp \\ \phi_{receive} = (y_1, \ldots, y_m) \\ \phi_{broadcast} = x \\ w = \perp \end{array} \middle| x := f_\ell(y_1, \ldots, y_m) \right\}.$$

Protocol $\Pi_{YOSO-LHE}$: Input and Output

Input: *This step is run by an input role.*
To provide input x to committee C, the jth input role does the following:
- Computes a shamir sharing of its secret input x as $(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x; \rho_{I_i})$ with threshold t.
- Inputs $(\textsc{Send}, I_j, ((C_1, x_1, \perp), \ldots, (C_n, x_n, \perp)), \perp, (x, \rho_{I_i}))$ to $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$.

The ith role in C may then read her share of input x in the following round by inputting $(\textsc{Read}, C_i, I_j, 1)$ to $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$.

Decrypt: *This step is run by a committee C of size n, in round r.* To reveal her share, each member C_i of committee C does the following:
- Input $(\textsc{Send}, C_i, \perp, x_i, \perp)$ to $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$.

Let $\mathcal{Q} \subseteq [n]$ denote the indices of the roles in C who provided a valid share $x_i \neq \perp$ read from $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ using $(\textsc{Read}, C_i, r)$. Anyone can then reconstruct x as $x \leftarrow \mathsf{Rec}(\{x_i\}_{i \in \mathcal{Q}})$.

Output: *This step is run by committee C of size n.*
The committee C calls Decrypt using shares $(\mathsf{out}_1, \ldots, \mathsf{out}_n)$. [a]

[a]Note that the associated relation $\mathcal{R}_{\mathsf{Dec}}$ would involve the values that were used to compute out_i as a part of $\phi_{receive}$.

The additions in the circuit can be done via local computation. However, multiplication gates still require interaction and thus require passing state over to a new committee. To this end we introduce two new committees:

- $M_{l,1}, \ldots, M_{l,n}$ denotes the roles of the committee (of size n) responsible for the lth multiplication.
- $O_{l,1}, \ldots, O_{l,n}$ 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.)

The multiplication committee uses the decryption operation to reveal intermediate values, avoiding the need for any new relations. We start by assuming that a sharing of a beaver triple is held by our multiplication and output committee, and will subsequently show how such a triple may be produced.

Protocol $\Pi_{YOSO-LHE}$: Evaluation

Add: *This step is run by a committee C of size n.*
To add $(x_1, \ldots, x_n)$ (representing shares of x) and $(y_1, \ldots, y_n)$ (representing shares of y), the ith member of the committee C does the following.

- Compute her share of the sum as $z_i = x_i + y_i$.

Mult: *This step is run by committee M_l of size n and O_l of size n*
To multiply values

- $(x_{l,1}, \ldots, x_{l,n})$ (representing shares of x_l) and
- $(y_{l,1}, \ldots, y_{l,n})$ (representing shares of y_l)

(where the sharing is held by committee M_l) using the Beaver triple

- $(a_{l,1}, \ldots, a_{l,n})$, $(a'_{l,1}, \ldots, a'_{l,n})$,
- $(b_{l,1}, \ldots, b_{l,n})$, $(b'_{l,1}, \ldots, b'_{l,n})$
  (where the sharing is held by committees M_l and O_l, respectively) and
- $(c'_{l,1}, \ldots, c'_{l,n})$, (where the sharing is held by committee O_l),

The ith member of the committee M_l does the following to compute her share of $\epsilon_l = a_l - x_l$ and $\delta_l = b_l - y_l$.

- Compute the difference $\epsilon_{l,i} = a_{l,i} - x_{l,i}$.
- Compute the difference $\delta_{l,i} = b_{l,i} - y_{l,i}$.

The committee M_l then calls Decrypt using shares $(\epsilon_{l,1}, \ldots, \epsilon_{l,n})$ and $(\delta_{l,1}, \ldots, \delta_{l,n})$ to reconstruct the values ϵ_l and δ_l. [a]
The ith member of the committee O_l does the following to compute her share of $z_l = x_l y_l$.

- Reconstruct ϵ and δ by inputing $(\text{READ}, M_{l,i}, l+2)$ to $\mathcal{F}^{\mathsf{hom}}_{\mathsf{VSP}}$ (as described in Decrypt) for each ith member of M_l. Note that the multiplication gate at depth l is evaluated in round $l+2$.
- Compute $z_{l,i} := c'_{l,i} - \epsilon_l b'_{l,i} - \delta_l a'_{l,i} + \epsilon_l \delta_l$.

[a] Note that the associated relation $\mathcal{R}_{\mathsf{Dec}}$ would involve the set of received shares $a_{l,i}, b_{l,i}$ and the values that were used to compute $x_{l,i}, y_{l,i}$ as a part of $\phi_{receive}$.

To produce beaver triples we will need two final types of committee. Let $A_{l,1}, \ldots, A_{l,h}$ and $B_{l,1}, \ldots, B_{l,h}$ 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. Two additional relations are also needed to ensure the triples produced are well-formed:

$$\mathcal{R}_{\mathsf{Beaver,A}} = \left\{ \begin{array}{l|l} \phi_{send} = \begin{array}{l} \big((M_1, a_1, \perp), \ldots, (M_n, a_n, \perp), \\ (O_1, a'_1, \perp), \ldots, (O_n, a'_n, \perp)\big) \end{array} & \\ \phi_{receive} = \perp & (a_1, \ldots, a_n) \leftarrow \mathsf{Share}(a; \rho) \\ \phi_{broadcast} = \perp & \wedge\ (a'_1, \ldots, a'_n) \leftarrow \mathsf{Share}(a; \rho') \\ w = \big(a, \rho, \rho'\big) & \end{array} \right\},$$

The below relation for the roles in the helper committee B_l is required to check that the correct linear homomorphic function f_i is applied on the incoming messages (say $y_1, \ldots, y_m$) of a committee role $O_{l,i}$. For simplicity, we describe f_i as a tuple of coefficients $(c_1, \ldots, c_m, c)$ to represent the operation $f_i(y_1, \ldots, y_m) = c_1 y_1 + c_2 y_2 + \cdots + c_m y_m + c$.

$$\mathcal{R}_{\mathsf{Beaver,B}} = \left\{ \begin{array}{l|l} \phi_{send} = \begin{pmatrix}(M_1, b_1, \perp), \ldots, (M_n, b_n, \perp), \\ (O_1, b'_1, f_1), \ldots, (O_n, b'_n, f_n)\end{pmatrix} & (b_1, \ldots, b_n) \leftarrow \mathsf{Share}(b; \rho) \\ \phi_{receive} = \perp & \wedge\ (b'_1, \ldots, b'_n) \leftarrow \mathsf{Share}(b; \rho') \\ \phi_{broadcast} = \perp & \wedge(0_1, \ldots, 0_n) \leftarrow \mathsf{Share}(0; \rho'') \\ w = \big(b, \rho, \rho', \rho''\big) & \wedge f_i = (b, \ldots, b, 0_i) \text{ for } i \in [n] \end{array} \right\},$$

Protocol $\mathit{\Pi}_{YOSO-LHE}$: Beaver triple generation

MakeBeaver: *This step is run by two helper committees A_l and B_l of size h each*
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.
  - Inputs $\big(\textsc{Send}, A_{l,j}, \big((M_{l,1}, a_{l,j,1}, \perp), \ldots, (M_{l,n}, a_{l,j,n}, \perp), (O_{l,1}, a'_{l,j,1}, \perp), (O_{l,n}, a'_{l,j,n}, \perp)\big), \perp, (a_{l,j}, \rho_{l,j}, \rho'_{l,j})\big)$ to $\mathcal{F}^{\mathsf{hom}}_{\mathsf{VSP}}$.
- Let $\mathcal{Q}_{A_l} \subseteq [h]$ denote the indices of the roles whose Send verified successfully. Let $a_l = \sum_{j \in \mathcal{Q}_{A_l}} a_{l,j}$. Then,
  - Each role $M_{l,i}$ can retrieve her share of a_l as $a_{l,i} = \sum_{j \in \mathcal{Q}_{A_l}} a_{l,j,i}$, where $a_{l,j,i}$ was read from $\mathcal{F}^{\mathsf{hom}}_{\mathsf{VSP}}$ using $(\textsc{Read}, M_{l,i}, A_{l,j}, l+2)$.
  - Each role $O_{l,i}$ can retrieve her share of a_l as $a'_{l,i} = \sum_{j \in \mathcal{Q}_{A_l}} a'_{l,j,i}$, where $a'_{l,j,i}$ was read from $\mathcal{F}^{\mathsf{hom}}_{\mathsf{VSP}}$ using $(\textsc{Read}, O_{l,i}, A_{l,j}, l+2)$.
- 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.
  - Compute a zero sharing as $(0_{l,j,1}, \ldots, 0_{l,j,n}) \leftarrow \mathsf{Share}(0; \rho''_{l,j})$ with threshold t.
  - Set the function $f'_i = (b_{l,j}, \ldots, b_{l,j}, 0_{l,j,i})$ for each $i \in [n]$. Recall that this function would take as input $y(O_{l,i})$ comprising of the messages (say $y_1, \ldots, y_h$) received by $O_{l,i}$ from sender roles $A_{l,1}, \ldots, A_{l,h}$ and outputs $b_{l,j}y_1 + \cdots + b_{l,j}y_h + 0_{l,j,i}$, where any $\perp$ values are to be interpreted as 0. (The shares of 0 are used to ensure that the value $b_{l,j}$ is not leaked to $O_{l,i}$).
  - Input $\big(\textsc{Send}, B_{l,j}, \big((M_{l,1}, b_{l,j,1}, \perp), \ldots, (M_{l,n}, b_{l,j,n}, \perp), (O_{l,1}, b'_{l,j,1}, f'_1), \ldots, (O_{l,n}, b'_{l,j,n}, f'_n)\big), \perp, (b_{l,j}, \rho_{l,j}, \rho'_{l,j}, \rho''_{l,j})\big)$ to $\mathcal{F}^{\mathsf{hom}}_{\mathsf{VSP}}$.
- Let $\mathcal{Q}_{B_l} \subseteq [h]$ denote the indices of the roles whose Send verified successfully. Let $b_l = \sum_{j \in \mathcal{Q}_{B_l}} b_{l,j}$. Let $(b'_{l,j,i}, c'_{l,j,i})$ denote the value that $O_{l,i}$ receives from $\mathcal{F}^{\mathsf{hom}}_{\mathsf{VSP}}$ using $(\textsc{Read}, O_{l,i}, B_{l,j}, l+2)$. Then,
  - Each role $M_{l,i}$ can retrieve her share of b_l as $b_{l,i} = \sum_{j \in \mathcal{Q}_{B_l}} b_{l,j,i}$, where $b_{l,j,i}$ was read from $\mathcal{F}^{\mathsf{hom}}_{\mathsf{VSP}}$ using $(\textsc{Read}, M_{l,i}, B_{l,j}, l+2)$.
  - Each role $O_{l,i}$ can retrieve her share of b_l as $b'_{l,i} = \sum_{j \in \mathcal{Q}_{B_l}} b'_{l,j,i}$.
  - Each role $O_{l,i}$ can retrieve her share of $c_l = a_l b_l$ as $c'_{l,i} = \sum_{j \in \mathcal{Q}_{B_l}} c'_{l,j,i}$.

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 ($h < n$).

The proof of Theorem 3 may be found in Appendix C.4.

C Security proofs

C.1 Security of YaOSO

To prove the protocol $\mathit{\Pi}_{YaOSO}$ YOSO realises the functionality $\mathcal{F}_f$ with guaranteed output delivery, we will show that the protocol $\mathsf{YoS}(\mathit{\Pi}_{YaOSO})$ UC realises the functionality $\mathcal{F}_f$. We may do this by constructing a simulator $\mathcal{S}_{YaOSO}$ for any given adversary $\mathcal{A}$, such that:

$$REAL_{\mathsf{YoS}(\mathit{\Pi}_{YaOSO}),\mathcal{A},\mathcal{E}}(1^\kappa) \approx IDEAL_{\mathcal{F}_f,\mathcal{S}_{YaOSO},\mathcal{E}}(1^\kappa).$$

Proof. For a given adversary $\mathcal{A}$ we define the simulator $\mathcal{S}_{YaOSO}$. $\mathcal{S}_{YaOSO}$ internally uses the simulator (say $\mathcal{S}_{\mathsf{sm}}$) of the underlying semi-malicious protocol $\mathit{\Pi}_{\mathsf{sm}}$. The idea is for $\mathcal{S}_{YaOSO}$ to transform a malicious adversary in $\mathit{\Pi}_{YaOSO}$ to a semi-malicious adversary in $\mathit{\Pi}_{\mathsf{sm}}$. This is done by verifying the messages that the maliciously corrupt roles use to invoke $\mathcal{F}_{\mathsf{VSP}}$ (which $\mathcal{S}_{YaOSO}$ has access to). If the messages verify, they are forwarded to $\mathcal{S}_{\mathsf{sm}}$ on behalf of semi-malicious corrupt parties. Else, the messages are recomputed using default input and randomness. In this manner, the messages corresponding to the underlying protocol can be simulated. Additionally, $\mathcal{S}_{YaOSO}$ also invokes the simulator of the adaptive garbling scheme (Appendix A.2) to simulate the garbled circuits and labels of honest roles.

First, we argue regarding the correctness of the $\mathit{\Pi}_{YaOSO}$. The correctness of the garbling scheme and threshold secret sharing ensures that the second-round messages of $\mathit{\Pi}_{\mathsf{sm}}$ obtained via the evaluation of garbled circuits would be correct. Now, correctness of $\mathit{\Pi}_{YaOSO}$ follows directly from the correctness of $\mathit{\Pi}_{\mathsf{sm}}$.

Next, we define the simulator below. The set of indices corresponding to honest roles and corrupt roles in committee C are denoted as $\mathcal{H}_C$ and $\mathcal{I}_C$ respectively.

Simulator $\mathcal{S}_{YaOSO}$

Let $\mathcal{S}_{\mathsf{sm}}$ denote the simulator of the underlying semi-malicious protocol $\mathit{\Pi}_{\mathsf{sm}}$.

Input: For the input committee, the simulator:
- For each honest input role $j \in \mathcal{H}_I$,
  - Receive $\{\mathsf{msg}_j^1\}$ by interaction with $\mathcal{S}_{\mathsf{sm}}$.
  - Compute the simulated garbled circuit as $\mathtt{GC}_j \leftarrow \mathtt{simGC}(1^n, \phi(\mathtt{C}_j), \text{“}ckt\text{”})$, where $\phi(\mathtt{C}_j)$ denotes the topology of the circuit (that does not depend on the hard coded values).
  - When a corrupt role attempts to input $(\textsc{Read}, I_j, 1)$ to $\mathcal{F}_{\mathsf{VSP}}$, return the response $(\mathsf{msg}_j^1, \mathtt{GC}_j)$ as computed above.
  - When a corrupt computation committee role $i \in \mathcal{I}_E$ attempts to input $(\textsc{Read}, E_i, I_j, 1)$ to $\mathcal{F}_{\mathsf{VSP}}$, return as response a set of random shares $\{s_{j,l,i}^{(0)}, s_{j,l,i}^{(1)}\}_{l\in[\mathsf{L}]}$.
- On behalf of $\mathcal{F}_{\mathsf{VSP}}$, verify the $\textsc{Send}$ input by corrupt input roles $j \in \mathcal{I}_I$. Replace the shares with $\perp$ if the verification fails.
- When a corrupt computation committee role $i \in \mathcal{I}_E$ attempts to input $(\textsc{Read}, E_i, I_j, 1)$ to $\mathcal{F}_{\mathsf{VSP}}$ for $j \in \mathcal{I}_I$, return $\{s_{j,l,i}^{(0)}, s_{j,l,i}^{(1)}\}_{l\in[\mathsf{L}]}$.

At the conclusion of this round the simulator knows $(x_k, \{\mathsf{msg}_k^1\}, \{s_{k,l,i}^{(0)}, s_{k,l,i}^{(1)}\}_{l\in[\mathsf{L}],i\in[n]})$ (consider default values in case a corrupt role aborts or the verification fails) for each corrupt input role $k \in \mathcal{I}_I$ as they are leaked by $\mathcal{F}_{\mathsf{VSP}}$.

The simulator may provide these inputs $\{x_k\}_{k\in\mathcal{I}_I}$ to the ideal functionality to receive $\mathsf{out} = f(x_1,\ldots,x_m)$. This out is provided to $\mathcal{S}_{\mathsf{sm}}$ as the response from its ideal functionality when invoked by $\mathcal{S}_{\mathsf{sm}}$.

Computation: For the computation committee the simulator:

- Interacts with $\mathcal{S}_{\mathsf{sm}}$ as follows: Send the first-round message msg_k^1 on behalf of corrupt roles $k \in \mathcal{I}_I$. Receive the second round messages msg_j^2 corresponding to the honest input roles $j \in \mathcal{H}_I$.
- For each $j \in \mathcal{H}_I$
  - Compute the set of simulated garbled labels corresponding to $\mathtt{GC}_j$ as $(K_{j,1},\ldots,K_{j,\mathsf{L}}) \leftarrow \mathtt{simGC}(1^n, \mathsf{msg}_j^2, \text{“}\mathit{input}\text{”})$
  - Compute the shares $\{s_{j,l,i}^{(b_l)}\}_{i\in\mathcal{H}_E} \leftarrow \mathsf{SH.SimShare}(\{s_{j,l,i}^{(b_l)}\}_{i\in\mathcal{I}_E}, K_{j,l})$ for $l \in [\mathsf{L}]$ (where $b_1,\ldots,b_{\mathsf{L}} = \mathsf{msg}_1^1||\ldots\mathsf{msg}_m^1$).
- When a corrupt role attempts to input $(\textsc{Read}, E_i, 2)$ for $i \in \mathcal{H}_E$, return as response $\{s_{j,l,i}^{(b_l)}\}_{j\in I, l\in[\mathsf{L}]}$ as computed above for $j \in \mathcal{H}_I$ and leaked via $\mathcal{F}_{\mathsf{VSP}}$ for $j \in \mathcal{I}_I$.

We prove the indistinguishability of the real and ideal world through a series of hybrids.

Real $H0$**:** Run everything as in the real protocol, using the honest roles inputs. Note, through the use of $\mathcal{F}_{\mathsf{VSP}}$ the inputs of corrupt roles will already be known to the simulator at this point, allowing them to be input the ideal functionality to receive $\mathsf{out} = f(x_1,\ldots,x_m)$.

Hybrid $H1$ **(Simulate honest shares):** The honest threshold shares held by $i \in \mathcal{H}_E$ corresponding to labels of $\mathtt{GC}_j$ of honest input roles $j \in \mathcal{H}_I$ are set as $\{s_{j,l,i}^{(b_l)}\}_{i\in\mathcal{H}_E} \leftarrow \mathsf{SH.SimShare}(\{s_{j,l,i}^{(b_l)}\}_{i\in\mathcal{I}_E}, K_{j,l})$ for $l \in [\mathsf{L}]$ (where $b_1,\ldots,b_{\mathsf{L}} = \mathsf{msg}_1^1||\ldots\mathsf{msg}_m^1$).

Hybrid $H2$ **(Simulate garbled circuits of honest input roles):** The garbled circuit and corresponding labels of honest input role $j \in \mathcal{H}_I$ are computed as $\mathtt{GC}_j \leftarrow \mathtt{simGC}(1^n, \phi(\mathtt{C}_j), \text{“}\mathit{ckt}\text{”})$ and $(K_{j,1},\ldots,K_{j,\mathsf{L}}) \leftarrow \mathtt{simGC}(1^n, \mathsf{msg}_j^2, \text{“}\mathit{input}\text{”})$.

Hybrid $H3$ **(Simulate the messages of** Π_{sm}**):** The first and second round messages of the underlying protocol Π_{sm} i.e. msg_j^1 and msg_j^2 for $j \in \mathcal{H}_I$ are obtained via the simulator $\mathcal{S}_{\mathsf{sm}}$. At this point the simulator no longer needs access to honest party inputs.

We show that the hybrids in our sequence are indistinguishable.

$H0 \approx H1$ The indistinguishablility of these hybrids follows from the share simulatability of the threshold secret sharing scheme (Appendix A.1).

$H1 \approx H2$ Indistinguishability of $H1$ and $H2$ follows via reduction to the adaptive privacy of the garbling scheme (Appendix A.2).

$H2 \approx H3$ Indistinguishability of $H2$ and $H3$ follows from semi-malicious security of π_{sm}.

C.2 Security of YOSO-GLS

We will now prove security of the $\Pi_{YOSO-GLS}$ protocol (Theorem 2).

Proof. To prove the protocol $\Pi_{YOSO-GLS}$ YOSO realises the functionality $\mathcal{F}_f$ with guaranteed output delivery, we will show that the protocol $\mathsf{YoS}(\Pi_{YOSO-GLS})$ UC realises the functionality $\mathcal{F}_f$. We may do this by constructing a simulator $\mathcal{S}$ for any given adversary $\mathcal{A}$, such that:

$$REAL_{\mathsf{YoS}(\Pi_{YOSO-GLS}),\mathcal{A},\mathcal{E}}(1^\kappa) \approx IDEAL_{\mathcal{F}_f,\mathcal{S},\mathcal{E}}(1^\kappa).$$

For a given adversary $\mathcal{A}$ we define the simulator as follows:

Simulator $\mathcal{S}$

Begin by reading public parameters leaked from $\mathcal{F}_{\mathsf{Setup}}$. Allow the adversary to control the corrupt roles, simulating the honest roles and $\mathcal{F}_{\mathsf{VSP}}$

KGen: For honest roles K_i perform key generation as described in the protocol. At the conclusion of this round the simulator knows $sk_j = (sk_{j,1}, \dots, sk_{j,n})$ for each corrupt role K_j $(j \in \mathcal{I}_K)$ as they are leaked by $\mathcal{F}_{\mathsf{VSP}}$.

Input: Rather than encrypting their input, each role may instead encrypt 0 under the TFHE keys to get $C_i \leftarrow \mathsf{Enc}(\{pk_i\}_{i\in\mathcal{K}}, 0; \rho_{I_i})$ which may then be input to $\mathcal{F}_{\mathsf{VSP}}$ as $(\textsc{Send}, I_i, \perp, C_i, (0, \rho_{I_i}))$.
At the conclusion of this round the simulator knows input x_j for each corrupt input role I_j $(j \in \mathcal{I}_I)$. The simulator may provide these inputs to the ideal functionality to receive $\mathsf{out} = f(x_1, \dots, x_m)$

Computation: For the computation committee the simulator:
- Homomorphically derives the ciphertext C according to the protocol
- Derives partial decryptions for corrupt roles in the computation committee as $d_i \leftarrow \mathsf{PDec}(\{pk_k\}_{k\in\mathcal{K}}, csk_i, C)$ for $csk_i = \{sk_{j,i}\}_{j\in K}$.
- The simulator then produces partial decryptions for the honest roles $\{d_j\}_{j\in[n]\setminus\mathcal{I}_{\mathsf{Computation}}} \leftarrow \mathsf{SimPDec}(C, \{pk_i\}_{i\in\mathcal{K}}, \{sk_i\}_{i\in\mathcal{I}_K}, \{(csk_j, d_j)\}_{j\in\mathcal{I}_E}, \mathsf{out})$
- When a corrupt role attempts to input $(\textsc{Read}, E_j, 3, \mathcal{R}_{\mathsf{Eval}})$ to $\mathcal{F}_{\mathsf{VSP}}$ for an honest role E_j, replace the response with d_j as computed above.

We prove the indistinguishability of the real and ideal world through a series of hybrids.

Real $H0$**:** Run everything as in the real protocol, using the honest roles inputs. Note, through the use of $\mathcal{F}_{\mathsf{VSP}}$ the inputs of corrupt roles will already be known to the simulator at this point, allowing them to be input the ideal functionality to receive $\mathsf{out} = f(x_1, \dots, x_m)$.

Hybrid $H1$ **(Partial decryption simulation):** Simulate partial decryptions for honest roles as:

$$\{d_j\}_{j\in[n]\setminus\mathcal{I}_{\mathsf{Computation}}} \leftarrow \mathsf{SimPDec}(C, \{pk_i\}_{i\in\mathcal{K}}, \{sk_i\}_{i\in\mathcal{I}_K}, \{(csk_j, d_j)\}_{j\in\mathcal{I}_E}, \mathsf{out}),$$

where desired output out should be the value returned by the ideal functionality. These partial decryptions should then be returned whenever a corrupt role inputs $(\textsc{Read}, E_j, 3)$ to $\mathcal{F}_{\mathsf{VSP}}$ for $j \in \mathcal{H}_E$.
Note, this requires access to secret keys sk_i for corrupt key generation roles $i \in \mathcal{I}_K$, as well as partial decryptions d_j and computation keys csk_j for $j \in \mathcal{I}_E$. The simulator has these secrets as they are leaked by $\mathcal{F}_{\mathsf{VSP}}$, and may use them to deterministically compute partial decryptions. These partial decryptions should then be returned when a corrupt role inputs $(\textsc{Read}, E_j, 3)$ for $j \in \mathcal{H}_E$.

Hybrid $H2$ (Encrypt 0 for honest roles): Replace encryptions of inputs from honest roles by encryptions of 0. At this point the simulator no longer needs access to honest role inputs.

First, we will prove correctness of the real protocol. The $\mathcal{F}_{\mathsf{Setup}}$ functionality ensures correct sampling of the public parameters, while commnication through the $\mathcal{F}_{\mathsf{VSP}}$ functionality with $\mathcal{R}_{\mathsf{KGen}}$ enforces that all public keys used when encrypting are well-formed. All input ciphertexts are ensured to be encryptions under some randomness by $\mathcal{F}_{\mathsf{VSP}}$ with $\mathcal{R}_{\mathsf{Enc}}$. As a result partial decryptions under the correct keys, as enforced by $\mathcal{F}_{\mathsf{VSP}}$ with $\mathcal{R}_{\mathsf{Eval}}$, will recombine to produce $f(x_1, \ldots, x_m)$ following from correctness of the TFHE scheme (Definition 2). Note, we are guaranteed to have sufficient partial decryptions by the honest majority of the computation committee.

We will now prove that the hybrids in our sequence are indistinguishable.

$H0 \approx H1$ The indistinguishablility of these hybrids follows from the partial decryption simulatability of the TFHE scheme (Definition 4). An adversary successfully distinguishing $H0$ and $H1$ with non-negligible probability may be used to win the partial decryption simulatability game $\mathsf{Game}^{ParDecSim}_{\mathcal{A},n,d,f,\mathsf{TFHE}}(\kappa)$, with the same probability. This may be done by generating secret keys for honest roles through use of the $\mathcal{O}\mathsf{KGen}$ and registering all corrupt keys sent on $\mathcal{F}_{\mathsf{VSP}}$ with $\mathcal{O}\mathsf{KReg}$ using the randomness leaked to the simulator. The point-to-point messages read by corrupt roles E_i for $i \in \mathcal{I}_E$, may be replaced by the simulator with key shares received by invoking $\mathcal{O}\mathsf{Corr}(i)$. By the threshold guarantees of the committees K and E, we are guaranteed that the $|\mathcal{H}_{\mathsf{KGen}}| \geq 1$ and $|\mathcal{I}_{\mathsf{Computation}}| \leq t$. The partial decryptions for honest roles in the protocol may then be set to the challenge provided in the game. Thus, if our adversary guesses $H0$ we may guess $b = 0$ in the game, guessing $b = 1$ otherwise.

$H1 \approx H2$ Indistinguishability of $H1$ and $H2$ follows by a reduction to the semantic security of the TFHE scheme (Definition 3). Encrytptions on behalf of honest input roles may be replaced one at a time, maintaining indistinguishability. An adversary successfully distinguishing these cases may then be used to win $\mathsf{Game}^{IND-CPA}_{\mathcal{A},n,t,\mathsf{TFHE}}(\kappa)$. We will again map corruptions in K to uses of the $\mathcal{O}\mathsf{KReg}$ and corruptions in E to uses of $\mathcal{O}\mathsf{Corr}$. This provides the same guarantees that $|\mathcal{H}_{\mathsf{KGen}}| \geq 1$ and $|\mathcal{I}_{\mathsf{Computation}}| \leq t$. When replacing the input of an honest input role I_i we may input messages $(0, x_i)$ to the game. The challenge may then be used as the ciphertext for I_i in the protocol. If the adversary guesses it is in the hybrid where the plaintext has been replaced, we guess $b = 0$ in the game, guessing $b = 1$ otherwise. This allows winning the game with the distinguishing advantage the adversary has on the hybrids.

C.3 Security of $\mathit{\Pi}_{\mathsf{Setup}}$

Proof. We will show that $\mathsf{YoS}(\mathit{\Pi}_{\mathsf{Setup}})$ UC realises $\mathcal{F}^{\mathsf{TFHE}}_{\mathsf{Setup}}$ in the $\mathcal{F}_{\mathsf{VSP}}$-hybrid model. For a real world adversary $\mathcal{A}$ we will construct ideal world adversary $\mathcal{S}$

such that the real and ideal ensembles are indistinguishable, i.e.

$$REAL_{\mathsf{YoS}(\Pi_{\mathsf{Setup}}),\mathcal{A},\mathcal{E}}(1^\kappa) \approx IDEAL_{\mathcal{F}^{\mathsf{TFHE}}_{\mathsf{Setup}},\mathcal{S},\mathcal{E}}(1^\kappa).$$

Simulator $\mathcal{S}$

The ideal functionality $\mathcal{F}^{\mathsf{TFHE}}_{\mathsf{Setup}}$ leaks the chosen matrix $\mathbf{B}$ to the simulator.

Sample: Run all honest roles in the sample committee as prescribed by the protocol.

Combine: The simulator ensures shares will reconstruct to $\mathbf{B}$.
- For each corrupt role $i \in \mathcal{I}_C$ compute $\mathbf{B}_i$ as in the protocol.
- Simulate $\mathbf{B}_i$ for $i \in \mathcal{H}_C$, conditioned on corrupt shares, such that they reconstruct to $\mathbf{B}$.

Shares sent to corrupt roles $\mathcal{F}_{\mathsf{VSP}}$ by honest roles in the sample committee are identically distributed in the real and ideal worlds. Any set of fewer than t shares is independent of the secret shared value. Therefore, simulated shares will be distributed identically to real shares, as there is at least one honest role in the sample committee.

C.4 Security of YOSO-LHE

To prove the protocol $\Pi_{YOSO-LHE}$ YOSO realises the functionality $\mathcal{F}_f$ with guaranteed output delivery, we will show that the protocol $\mathsf{YoS}(\Pi_{YOSO-LHE})$ UC realises the functionality $\mathcal{F}_f$. We may do this by constructing a simulator $\mathcal{S}$ for any given adversary $\mathcal{A}$, such that:

$$REAL_{\mathsf{YoS}(\Pi_{YOSO-LHE}),\mathcal{A},\mathcal{E}}(1^\kappa) \approx IDEAL_{\mathcal{F}_f,\mathcal{S},\mathcal{E}}(1^\kappa).$$

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

Lemma 3 (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.

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

Add Perfect correctness of the Shamir secret sharing 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.

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 $\mathcal{S}$

Input: For the input committee, the simulator:
- When a corrupt role C_i attempts to input $(\textsc{Read}, C_i, I_j, 1)$ to $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ for honest input role $j \in \mathcal{H}_I$, return $(\tilde{x}_{j,i}, \perp)$ where $\tilde{x}_{j,i}$ is a random share.
- On behalf of $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$, verify the $\textsc{Send}$ input by corrupt input roles $j \in \mathcal{I}_I$ and store the shares $(x_{j,1}, \ldots, x_{j,n})$. Replace the shares with $\perp$ if the verification fails.
- When a corrupt role C_i attempts to input $(\textsc{Read}, C_i, I_j, 1)$ to $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ for corrupt input role I_j, return $(x_{j,i})$.

At the conclusion of this round the simulator knows the input (x_k) (consider default values in case a corrupt role aborts or the verification fails) for each corrupt input role $k \in \mathcal{I}_I$ as they are leaked by $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$. The simulator may provide these inputs $\{x_k\}_{k\in\mathcal{I}_I}$ to the ideal functionality to receive $\mathsf{out} = f(x_1, \ldots, x_m)$.

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 honest role shares.* Let $\{x_i\}_{i\in\mathcal{I}_C}$ denote the shares held by corrupt roles in C (which the simulator knows because these can be deduced using the values leaked via the $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$). Compute the shares on behalf of honest roles such that they would be consistent with x as follows: Compute $\{x'_i\}_{i\in\mathcal{H}_C} \leftarrow \mathsf{SH.SimShare}(\{x_i\}_{i\in\mathcal{I}_C}, x)$.

Add: $\mathcal{S}$ uses the values learned earlier 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 with respect to the A_l helper committee:
   - When a corrupt role $M_{l,i}$ attempts to input $(\textsc{Read}, M_{l,i}, A_{l,j}, l+2)$ to $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ for $j \in \mathcal{H}_{A_l}$, return $(a_{l,j,i})$ where $a_{l,j,i}$ is set as a random share.
   - When a corrupt role $O_{l,i}$ attempts to input $(\textsc{Read}, O_{l,i}, A_{l,j}, l+2)$ to $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ for $j \in \mathcal{H}_{A_l}$, return $(a'_{l,j,i})$ where $a'_{l,j,i}$ is set as a random share.
   - On behalf of $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$, verify the $\textsc{Send}$ input by corrupt helper roles $j \in \mathcal{I}_{A_l}$ and store the shares $\{a_{l,j,i}, a'_{l,j,i}\}_{i\in[n]}$. Replace the shares with $\perp$ if the verification fails. Return the relevant share as response on behalf of $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ when corrupt roles in M_l and O_l invoke the $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ with $\textsc{Read}$ with respect to $A_{l,j}$.
2. With respect to the B_l helper committee, the $\mathcal{S}$ executes steps similar to the above (for values $\{b_{l,j,i}, b'_{l,j,i}\}_{i\in[n]}$), except that the $c'_{l,j,i}$ values also need to be simulated in a similar manner (i.e. set to random shares when honest roles in B_l are involved and as per the protocol when corrupt roles in B_l are involved).
3. Using the leakage from $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ and the shares sent on behalf of honest roles, the simulator can deduce the shares of a_l, b_l held by the corrupt roles in M_l and O_l and 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 values learned earlier.
2. Execute the simulation steps in $\mathsf{Decrypt}$ 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 values learned earlier.

Output: Using leaked values from $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$, $\mathcal{S}$ deduces the shares of the output 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 $H0$: The simulator does everything as in the real protocol. $\mathcal{F}_{\mathsf{VSP}}$ leaks inputs from corrupt roles to the simulator, allowing these to be input to the ideal functionality to receive the output $\mathsf{out} = f(x_1, \ldots, x_m)$.

Hybrid $H1$ (Simulate output shares): Replace honest shares broadcast through $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ in final call to the Decrypt procedure with simulated shares produced as $\{\mathsf{out}_i\}_{i \in \mathcal{H}_C} \leftarrow \mathsf{SimShare}(\{\mathsf{out}_i\}_{i \in \mathcal{I}_C}, \mathsf{out})$.

Hybrid $H2$ (Pick ϵ, δ Randomly, Simulate Honest Roles' Shares) When multiplication committee M_l decrypts ϵ and δ choose ϵ and δ uniformly at random and simulate honest shares:
- $\{\epsilon_{l,i}\}_{i \in \mathcal{H}_C} \leftarrow \mathsf{SimShare}(\{\epsilon_{l,i}\}_{i \in \mathcal{I}_C}, \epsilon)$,
- $\{\delta_{l,i}\}_{i \in \mathcal{H}_C} \leftarrow \mathsf{SimShare}(\{\delta_{l,i}\}_{i \in \mathcal{I}_C}, \delta)$.

Ideal $H3$ (Replace shares of honest inputs with random) Rather than sharing honest inputs, provide random shares to corrupt roles when they input $(\textsc{Read}, I_i, C_j, 1)$ to $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$.

We now prove that each pair in our sequence of hybrids is indistinguishable.

$H0 \approx H1$ We will prove indistinguishablility from the real world through an invariant over the evaluation of the circuit. The shares of honest inputs held by honest roles make up a uniform sharing conditioned on the shares of corrupt roles. For simplicity, we assume our circuit contains at least one multiplication gate. After the first multiplication the corresponding held by honest roles are independent of the view of the adversary. The process for beaver triple generation ensures this as both the A and B committees contain at least one honest role, resulting in the sum of their produced shares making up a uniformly random sharing of a uniform value, conditioned on corrupt shares. The honest shares of ϵ and δ are defined as $\epsilon_{l,i} = a_{l,i} - x_{l,i}$ and $\delta_{l,i} = b_{l,i} - y_{l,i}$ for $i \in \mathcal{H}_M$. Both $a_{l,i}$ and $b_{l,i}$ are uniform shares conditioned on the shares of the adversary, rendering the same true for $\epsilon_{l,i}$ and $\delta_{l,i}$. As $c'_{l,i}$ is produced through the use of the homomorphism of $\mathcal{F}_{\mathsf{VSP}}^{\mathsf{hom}}$ with an added fresh sharing of 0 the result of $c'_{l,i} - \epsilon_l b'_{l,i} - \delta_l a'_{l,i} + \epsilon_l \delta_l$ for $i \in \mathcal{H}_O$ will be similarly uniform shares conditioned on the shares of corrupt roles. Addition requires no communication, and produces new shares which are again distributed as part of a uniformly chosen sharing conditioned on the shares of the adversary. When decrypting for the final gate the shares for the honest roles may therefore instead be simulated by invoking share simulatability, here the output of the functionality may be used relying on the correctness of the protocol.

$H1 \approx H2$ As argued previously the values ϵ and δ are uniformly random, while the shares of these values held by the honest roles constitute a uniform sharing conditioned on the corrupt shares. Once again, the shares which are broadcasted by honest roles may be replaced by simulations, by invoking share simulatability.

$H2 \approx H3$ Finally, as the decryptions of honest shares throughout the circuit do not depend on honest shares of honest inputs the simulator may simply choose uniform random shares to provide to the corrupt roles. This results in an identical distribution and eliminates the need for the inputs of honest roles.