Statistical Layered MPC

Giovanni Deligios
ETH Zurich
gdeligios@ethz.ch

Anders Konring
Espresso Systems
anders.konring@gmail.com

Chen-Da Liu-Zhang
Lucerne University of Applied Sciences and Arts & Web3 Foundation
chendaliu@gmail.com

Varun Narayanan[*]
University of California, Los Angeles
varunnkv@gmail.com

Abstract

The seminal work of Rabin and Ben-Or (STOC'89) showed that the problem of secure n-party computation can be solved for $t < n/2$ corruptions with guaranteed output delivery and statistical security. This holds in the traditional static model where the set of parties is fixed throughout the entire protocol execution.

The need to better capture the dynamics of large scale and long-lived computations, where compromised parties may recover and the set of parties can change over time, has sparked renewed interest in the proactive security model by Ostrovsky and Yung (PODC'91). This abstraction, where the adversary may periodically uncorrupt and corrupt a new set of parties, is taken even a step further in the more recent YOSO and Fluid MPC models (CRYPTO'21) which allow, in addition, disjoint sets of parties participating in each round. Previous solutions with guaranteed output delivery and statistical security only tolerate $t < n/3$ corruptions, or assume a random corruption pattern plus non-standard communication models. Matching the Rabin and Ben-Or bound in these settings remains an open problem.

In this work, we settle this question considering the unifying Layered MPC abstraction recently introduced by David et al. (CRYPTO'23). In this model, the interaction pattern is defined by a layered acyclic graph, where each party sends secret messages and broadcast messages only to parties in the very next layer. We complete the feasibility landscape of layered MPC, by extending the Rabin and Ben-Or result to this setting. Our results imply maximally-proactive MPC with statistical security in the honest-majority setting.

[*]V. Narayanan was suported by NSF grants CNS-2246355, CCF-2220450, US-Israel BSF grant 2022370, and by Sunday Group.

Contents

1 Introduction

1.1 Setting

In the problem of secure multi-party computation (MPC) [36, 20, 3, 8, 33] a set of mutually distrusting parties jointly computes a function of their private data, so that nothing about their data beyond the output of the function is leaked.

MPC protocols are traditionally designed assuming a *static* set of n parties, that are required to stay online throughout the entire protocol execution. Security is guaranteed as long as up to a threshold of parties are compromised at any point throughout the entire protocol execution. In this setting, and assuming bilateral secure channels, Ben-Or, Goldwasser and Wigderson [3] and Chaum, Crepeau and Damgard [9] showed protocols to compute *any* function with *perfect* security against computationally unbounded adversaries corrupting less than $n/3$ of the participants. Rabin and Ben-Or [33] showed that if one assumes ideal broadcast and settles for statistical (security up to some error probability) rather than perfect security, the resilience can be improved to the optimal threshold of less than $n/2$.

However, this static-participant model is ill suited for capturing large-scale secure computations. These can be extremely long-lived, making the assumption that servers remain online throughout the protocol hard to satisfy. For longer applications, it also seems reasonable to assume that parties that are compromised at some point in the protocol execution may be able to recover, especially if the corruption capability of the adversary is tied to some scarce resource. The gap between the reach of traditional models and the needs of some use-cases has revived interest in the mobile adversary model by Ostrovsky and Yung [32]. Here, the adversary can corrupt different sets of parties in each round, as long as the number of per-round corruptions does not exceed a fixed threshold.

Recently, models have been proposed which in addition address the need for a more dynamic participation of parties: the YOSO [19] and Fluid MPC [11] models. They take the mobile-adversary model a step further by allowing a different set of n parties (called a committee) to participate in each round of the computation.

The paper introducing the YOSO model [19] also presented a protocol with statistical security and guaranteed output delivery against a dishonest minority. However, this protocol *assumes* ideal secure channels (with strong properties) between parties that come online at different times, and the adversary corrupts parties independently and with a constant probability $\tau < 1/2$. The Fluid model [11] assumes a worst-case corruption model instead, where the adversary can choose to corrupt any $t < n/2$ out of the n committee members in each round. In this case, the original paper only provides protocols that fall short of full security (guaranteed output delivery) and achieve the weaker notion of security with abort instead. Recently in [13], the authors showed a protocol for *perfect* security and guaranteed output delivery, but tolerating only up to $t < n/3$ corruptions. In this work, we investigate this model further in the statistical setting striving for the optimal resilience of $t < n/2$ corruptions. More concretely, we ask the following question:

> *Is it possible to solve the MPC problem with statistical security and optimal resiliency of $n/2$ corruptions in the setting of dynamic committees?*

1.2 Contributions

We answer the question in the affirmative. We construct the first MPC protocol with dynamic committees achieving guaranteed output delivery and statistical security tolerating up to $t < n/2$ corruptions per round. We remark that prior works in this setting only tolerate $t < n/3$ corruptions or assume non-standard communication and adversary models. See Table 1 for a comparison and Section 1.3 for a discussion on related work.

We consider the *Layered MPC* setting that was recently introduced in [13] (this is equivalent to Fluid MPC with maximal-fluidity [11]). This model provides a clean and elegant framework to treat MPC with dynamic committees. The parties are situated in a layered communication network which accommodates n parties in each layer. A party in any layer has access to secure unilateral communication channels that allow them to securely communicate to each party in the very next layer. Additionally, the party can securely broadcast messages to all the parties in the very next layer. Thus, the parties in any layer of the network can only receive private or broadcast messages from the parties in the previous layer, and send private or broadcast messages to the parties in the very next layer. The adversary is restricted to corrupting a subset of up to t out of n parties in each layer, a specialization of the notion of general adversary structure [22]. We consider the problem of realizing MPC over a layered network that allows the parties in the final layer of the network to securely compute a pre-agreed function of the inputs provided by the parties in the first layer. We prove the following theorem.

Theorem 1. *Let f be an n-party function computed by an arithmetic circuit C of depth d over a finite field $\mathbb{F}$. Then, for any $t < n/2$, there is a polynomial-time and polynomial-communication (in n and $|C|$) layered protocol over a $O(d)$-layered network that computes f with statistical t-security.*

This result finally settles the feasibility landscape of optimally-resilient MPC with guaranteed output delivery in the layered model, complementing analogous results in the settings of perfect and computational security. Furthermore, as proven in ([13], Lemma 1), layered MPC implies maximally proactive MPC ([13], Definition 3), so that our result also implies the existence of a statistically-secure maximally-proactive MPC protocol in the honest majority setting for a *fixed* set of n parties.

Corollary 1. *Let f be an n-party function computed by an arithmetic circuit over a finite field $\mathbb{F}$ with depth d. Then, for any $t < n/2$, there is a t-resilient polynomial-time and polynomial-communication maximally proactive MPC protocol computing f in $O(d)$ rounds with statistical security.*

We provide explicit ideal functionalities for all the primitives we build, and formally reduce the security of our protocols to the security of their most basic building blocks (like linear information-theoretic message authentication codes), whose security is captured through formal definitions. Considering the significant complexity overhead that by design affects constructions with dynamic committees, we consider this modelling effort a contribution in its own right.

1.3 Related Work

Several works use the idea of designing player-replaceable protocols that operate with dynamically chosen committees [10, 5, 21] as a way to achieve adaptive security and/or improve the communication pattern. A broad spectrum of models (*e.g.*, [19, 11]) and variants thereof [34, 1] have been proposed to accommodate for the notion that may be summarized as *multi-party computation with dynamic committees.*

We are inspired by the latest work of [13] presenting the concept of layered MPC and showing that it is feasible to design MPC protocols that are secure in the perfect setting against an adversary that may corrupt t parties in each layer and presented implications for perfect MPC in related models. Also, they provided a result in the computational setting for $t < n/2$ but left the question of optimal corruption threshold in the statistical setting largely unexplored.

Other models in the realm of MPC with dynamic committees have considered the statistical setting:

Fluid MPC [11]. The original work of Fluid MPC presented a protocol secure in statistical setting with optimal corruption threshold but did not obtain guaranteed output delivery. Concretely, layered MPC can be viewed as a bare-bones version of the model of *maximally* fluid MPC, ignoring some of the desirable add-on features that are part of the fluid MPC model (*e.g.*, dynamic sampling of parties). As a result of this simplification, layered MPC does not need a new definition but can be cast as an instance of standard MPC with specialized adversary structure and interaction pattern.

YOSO MPC [19]. The authors introducing YOSO MPC also presented a protocol in the statistical setting (*aka.* IT-YOSO) with optimal corruption threshold $t < n/2$. But certain assumptions in the YOSO MPC model make techniques used in IT-YOSO incompatible with designing protocols in the layered setting. A concrete example is the assumption of access to idealized communication channels through the role assignment functionality. This functionality guarantees that messages sent through these channels will arrive at the recipient in an arbitrary future round. Thus, in contrast to layered MPC, a dishonest sender cannot change the message while it is traveling to the recipient and is, effectively, committed to its input. Another example, is the weaker[1] probabilistic adversary which changes how protocols can be designed securely. Consider the VSS protocol in IT-YOSO which relies on a $\mathcal{F}_{\mathsf{UPBeacon}}$ functionality to provide unpredictable randomness. Their implementation of $\mathcal{F}_{\mathsf{UPBeacon}}$ (Sec. 3.11, [19]) has k roles commit (WSS) to an L-bit string, then opens them in sequence and computes the sum of the correctly opened values. Transforming this protocol directly into the layered setting (with worst-case corruption) results in a large bias on the output. Finally, we note that while the techniques in IT-YOSO rely heavily on the properties of role assignment, it is not known how to realize role assignment without computational assumptions (last

[1]The term "weak" is to emphasize the difference between random point-corruptions and worst-case corruptions *within* committees. In reality, the adversary models are not comparable since only in the YOSO model does the protocol have the responsibility of sampling committees with the right distribution.

Maximally Proactive MPC with Dynamic Committees				
Functionality	Reference	Level	Security	Threshold
VSS	[13]	perfect	full	$t < n/3$
	[4]	computational	full	$t < n/4^*$
	[19] (YOSO)	statistical	full (w/setup†)	$t < n/2^*$
	This work	statistical	full	$t < n/2$
MPC	[32]	perfect	full	$t < n/d$
	[13]	perfect	full	$t < n/3$
	[13]	computational	full	$t < n/2$
	[19] (YOSO)	statistical	full (w/setup†)	$t < n/2^*$
	[11] (Fluid)	statistical	w/abort	$t < n/2$
	This work	statistical	full	$t < n/2$

Table 1: Protocols realizing primitives in the most extreme proactive settings. (*protocol security relies on the adversary only doing probabilistic corruption, †assumes access to ideal target-anonymous channels for future messaging)

paragraph - Sec. 1.2.2, [19]). A variant of YOSO, called *YOSO with worst-case corruptions* was recently considered in [31, 27] for the concrete problem of randomness generation. This variant considers a setting where committees are executed in sequence, and the adversary can corrupt a total of t committees. The works consider the setting where each committee consists of one single party, and parties have access to committing channels to the future.

We summarize the landscape of relevant results in Table 1, an updated version of the summary from [13].

2 Technical Overview

2.1 Challenges

An approach that has repeatedly proven successful in the MPC literature is to transform protocols that provide security with abort into protocols achieving full security, or guaranteed output delivery. However, popular techniques such as player elimination or dispute control [23, 2] rely on detecting and excluding pairs of parties when at least one is known to be corrupted, and then restarting the protocol from a previous step. This method reduces the number of parties while maintaining the same corruption ratio. Unfortunately, these techniques are not applicable to the layered setting, where new committees consist of entirely different parties.

An alternative approach is to build upon sub-protocols, such as verifiable secret sharing, which tolerate malicious adversaries from the outset. However, the only information-theoretic protocols that provide guaranteed output delivery in the layered setting [13] are resilient against $t < n/3$ corruptions, and the techniques used fail in the honest-majority setting with $t \geq n/3$ corruptions. In particular, the efficient verifiable secret sharing scheme in [13] is based on the perfectly secure VSS from Gennaro et al. [17], tailored to the $t < n/3$ setting. Conversely, the computational protocol from [13] (see [14] for details) inherently utilizes linearly-homomorphic cryptographic commitments.

Our construction follows a blueprint first adapted to the setting of dynamic committees in the original YOSO paper [19] which constructs a protocol with guaranteed output delivery with less than half random corruptions in each committee (layer). Their construction, as well as ours, follows the well-known BGW paradigm. Initially, clients in the first layer distribute their inputs using a verifiable secret sharing scheme. Then, servers in subsequent layers process the circuit gate by gate; for each gate, servers in consecutive layers compute shares of the output wire from shares of the input wires of that gate. Ultimately, a layer of clients holds the output of the circuit.

However, adapting the YOSO protocol to the layered setting presents its own unique challenges. The protocol outlined in [19] relies on a couple of crucial assumptions. Firstly, it relies on ideal point-to-point *committing* channels to the future. These channels allow parties in any layer to transmit messages to parties in *any* subsequent layer. More importantly, when a corrupt party in $\mathcal{L}_0$ sends a message to a party in a later layer $\mathcal{L}_k$ for $k > 1$, the channel does not allow the sender to modify the message in a later layer $\mathcal{L}_{k'}$ where $k' > 0$. Such channels trivially solve issues arising from rushing and future causal dependency attacks, because a corrupted sender cannot adjust their message based on information acquired throughout the intermediate layers until $\mathcal{L}_k$. Addressing these attacks across all the primitives we construct (including our own constructions of channels to future committees, distributed information-theoretic signatures, distributed commitments) constitutes the primary challenge in the layered model.

Secondly, the YOSO model assumes that the adversary corrupts parties independently and at random. To illustrate why this model is weaker, consider an example where an honest party $P \in \mathcal{L}_0$ wants to distribute a secret s to the future. For this purpose, P generates k additive shares $s = s_1 + \cdots + s_k$ and transmits s_i to $P_i \in \mathcal{L}_1$. The probability of the adversary learning s in this scenario is 2^{-k}. In contrast, in our model, the adversary can simply corrupt the first k parties in $\mathcal{L}_1$ and learn the secret with probability 1.

Getting rid of these two strong assumptions requires new techniques. In this section we go through each of the building blocks for our layered MPC protocol, explaining the main technical challenges and the novel ideas employed to overcome them.

2.2 Robust Linear Secret Sharing

Our protocol follows the share-evaluate-reconstruct paradigm, so that naturally linear secret sharing serves as a basic building block. In the $t < n/3$ setting, even simple secret sharing schemes (think of Shamir sharing) achieve the important property of *robustness*: that is, if the dealer samples the shares honestly, even if an adversary modifies up to t out of n shares, the original secret can be correctly reconstructed. In our $t < n/2$ setting, robustness can also be achieved, but only up to some error probability.

We employ a simple secret sharing scheme that achieves robustness when up to $t < n/2$ are swapped *independently* from honest ones, is linear, and only requires a single round for reconstruction. The construction follows well-known techniques that date back to Rabin and Ben-Or or earlier, (also used in [15]) based on linear information theoretic message authentication codes (MAC) (see Section 3.2 for details). The secret m is shared as $(m_1, \ldots, m_n)$ using a plain t-out-of-n secret sharing scheme (such as classical Shamir [35]). Then, each share m'_i is authenticated using n MAC keys $(k_{1i}, \ldots, k_{ni})$ producing MAC tags $(t_{1i}, \ldots, t_{ni})$. The actual shares of the robust scheme are then defined as $m_i = (m'_i, (k_{i1}, \ldots, k_{in}), (t_{1i}, \ldots, t_{ni}))$, containing the non-robust share m'_i, the tags authenticating m'_i, as well as one key used to authenticate every other non-robust share $m'_j \neq m'_i$. An adversary holding up to t of the authenticated shares cannot forge MAC tags for the remaining $n - t$ keys, from which the robustness of the scheme follows. Hence, shares for which only up to t MAC checks succeed, can be dropped to ensure that only correct shares are used during reconstruction. This scheme is only linear if the keys for the MACs are sampled according to an appropriate distribution, and therefore it does *not* provide linearity among sharings performed by different dealers. See details in Section 3.3.

Tolerating attacks by a rushing adversary that can modify corrupt shares after having observed the honest ones is also important for later constructions, but achieving this guarantee is postponed until our future broadcast functionality.

2.3 Future Messaging

This primitive effectively establishes secure channels to future layers. More specifically, future messaging enables any sender from an initial layer, say layer $\mathcal{L}_0$, to transmit a message m to any receiver in layer $\mathcal{L}_k$ for $k > 0$. If the sender is honest, the view of an adversary corrupting up to $t < n/2$ parties in *all* intermediate layers is independent from m. It is important to note that this functionality is considerably weaker compared to the channels assumed in the YOSO model: if the sender is corrupted, the adversary can alter the message until the final layer $\mathcal{L}_k$. This is in contrast to the channels assumed in YOSO, where the adversary must fix the message at the time of transmission, independently from their view in later layers.

When sender and receiver are in adjacent layers (case $k = 1$), they can simply communicate via the provided point-to-point secure channels. When sender and receiver are separated by exactly one layer of parties (case $k = 2$) future messaging reduces to the problem of one-way secure message transmission (SMT) [16]. In this special case, an easy solution is for the sender to distribute a t-out-of-n robust secret sharing of m to layer $\mathcal{L}_1$, so that each party in this layer holds a distinct share. Then, each party in layer $\mathcal{L}_1$ forwards their share to the receiver in layer $\mathcal{L}_2$, who reconstructs the secret. In this last step, it is crucial that the reconstruction procedure of the secret sharing scheme is non-interactive. Since all communication happens between parties in adjacent layers, we can take advantage of the provided secure point-to-point channels. Note that, because the receiver is honest (we are not interested in providing any security guarantees for corrupted receivers), we are in the best-case scenario outlined above: the adversary can modify corrupt shares but only independently from the honest ones. Therefore, the receiver can recover the message m thanks to the robustness of the secret sharing scheme.

The general case ($k > 2$) is tackled recursively: the robust secret sharing of the message is distributed by the sender towards some intermediate layer $\mathcal{L}_{k'}$ for $0 < k' < k$. The shares are then forwarded by parties in layer $\mathcal{L}_{k'}$ to the receiver in layer $\mathcal{L}_k$. However, if $k' \geq 2$ or similarly if $k - k' \geq 2$, we cannot take advantage of provided point-to-point channels. To overcome this problem, each share is treated by the sender as a new message for a receiver in layer $\mathcal{L}_{k'}$, and the procedure is iterated in a recursive fashion, with the base of the iteration being resolved by making use of point-to-point channels between adjacent layers. Choosing $k' = \lfloor \frac{k}{2} \rfloor$ results in $O\big((C \cdot n)^{\log(k)}\big)$ communication complexity for a small constant C, so that as long as the sender and receiver are separated by a constant number of layers, as is the case in all our constructions, the protocol is efficient.

2.4 Future Broadcast

Our secret sharing scheme fails to provide robustness against a rushing adversary. This primitive achieves this, and in addition it provides an agreement guarantee (hence the name *broadcast*) when the sender (dealer) is corrupted: all receivers agree on the same value.

Specifically, future broadcast allows a party in layer $\mathcal{L}_0$ to send a message m securely to a set of recipients in layer $\mathcal{L}_k$. Each honest recipient agrees on *the same* message m', even if the sender is corrupted, and $m' = m$ when the sender is honest. Moreover, the primitive allows an auxiliary layer $\mathcal{L}_{k'}$ for $0 < k' < k$ to decide whether to deliver the message m or not. Future broadcast also guarantees linearity among messages from the same sender: if $\mathcal{L}_{k'}$ can deliver messages m_1 and m_2 from the same sender, it can also deliver any linear combination $am_1 + bm_2$ of these messages. For a single recipient, the difference from future messaging is that the auxiliary layer can decide whether to reveal the message or not.

In our construction, the dealer samples n independent robust sharings of m and provides them to $\mathcal{L}_{k'}$ using future messaging. Then, each of these states are reconstructed towards a distinct party in a buffer layer. Each party in the buffer layer finally broadcasts the value they have reconstructed, and a majority decision over all broadcast values is taken. Because the final decision is over public values, the agreement property follows easily. What is more, the honest majority in the buffer layer ensures robustness, as all honest parties in the buffer layer reconstruct the correct m if the sender is honest. Note that it is not enough to distribute a single sharing to $\mathcal{L}_{k'}$ and have every party in this layer broadcast their share, because a rushing adversary sees the shares broadcasted by honest parties before broadcasting corrupt shares. Our robust secret sharing scheme provides no guarantees in this case. However, it is easy to observe that if the n sharings of the message m are independent this rushing attack does not apply for an honest receiver in the buffer layer.

For context, in the non-layered model, our future broadcast protocol effectively realizes a robust linear secret sharing scheme secure against a rushing adversary, with a reconstruction procedure consisting of two communication rounds. We did not optimize the efficiency of our protocol: to share an m-bit secret, the resulting share size is $O(n \cdot m \cdot \kappa)$, where κ is the statistical security parameter. This is to be compared with the most efficient robust protocols in the non-layered setting [29], where the share size is $m + O(\kappa \cdot \log n(\log n + \log m))$. However, efficient constructions sacrifice the linearity properties crucial in MPC applications, and it is anyway unclear how to use them in our model, as there is no generic way to adapt their interactive reconstruction procedures to the layered setting. In contrast, our construction is linear, conceptually simple, and secure in the layered setting. Details can be found in Section 5.

2.5 Information Theoretic Signatures

The future messaging primitives described until now provide no guarantees when the sender is dishonest. They allow a dishonest sender to correlate their message with the information learned in the layers previous to (and including) the output layer. As a result, the sender is not committed to a message until the time of its delivery. Information theoretic signature (IT signature) takes the first step in the direction of limiting this freedom.

In an IT signature protocol, a sender $\mathbf{S} \in \mathcal{L}_0$ entrusts an intermediary $\mathbf{M} \in \mathcal{L}_k$ with a signed message which it can securely and verifiably reveal to receiver(s) $\mathbf{R}$ in $\mathcal{L}_{k''}$. When $\mathbf{S}$ is honest, the receivers will reject a corrupt $\mathbf{M}$ who attempts to reveal a value different from the sender's message. When $\mathbf{M}$ is honest, a corrupt $\mathbf{S}$ is committed to a message when the protocol reaches a so called auxiliary layer, in that, the states of parties in the auxiliary layer fix a message m such that receivers output m at the end of the protocol. The protocol provides *no guarantees* when both $\mathbf{S}$ and $\mathbf{M}$ are corrupt: adversary can choose the message based on its view until the receiving layer.

Our protocol uses the same blueprint as that of the implementation of IT signature in the YOSO model:

- The sender transfers the signed message to the intermediary.
- The intermediary verifies the validity of the signature with the help of the parties in the subsequent layers.
- If the signature is verified to be valid, the state held by the intermediary defines a message that honest receivers will accept. If the signature check fails, the sender is forced to reveal the actual message to the auxiliary layer, thereby committing to the message.
- The receivers accept the message if the signature is valid. Since the signed message is given only to intermediary, if the intermediary is honest the privacy of the sender's input is preserved until the message is revealed to the intended receiver.

S with input m computes MAC tags $t_{i,j} = \mathsf{Aut}(k_{i,j}, m)$ with respect to keys k_{ij} for $i \in [n]$ and $j \in [\kappa]$, where κ is a security parameter. The message along with the MAC tags effectively constitute a one time signature of the message which is privately communicated to $\mathbf{M} \in \mathcal{L}_k$. To enable the verification of the signature, $\mathbf{S}$ sends the keys $\{k_{i,j}\}_j$ to $\mathcal{L}_{k+1}$ so that $P_i \in \mathcal{L}_{k+1}$ receives $k_{i,j}$ for all $j \in [\kappa]$. We will refer to the parties in this layer as key-holders.

A corrupt sender may supply malformed signatures which can potentially sabotage the protocol by having the receivers reject an honest $\mathbf{M}$. To avoid this, $\mathbf{M}$ and the key-holders check the validity of the MACs provided by $\mathbf{S}$. This is achieved using cut and choose: each key-holder P_i announces a random subset $\mathrm{IND}_i \subseteq [\kappa]$ on which both P_i and the sender reveal their "versions" of $k_{i,j}$. The intermediary validates the MAC using the key revealed by the sender for each $j \in \mathrm{IND}_j$. If any of the checks fail, $\mathbf{M}$, now convinced that $\mathbf{S}$ is corrupt, complains forcing $\mathbf{S}$ to reveal m to parties in layer $\mathcal{L}_{k'}$. Otherwise, if some key-holder P_i's keys are different from the sender's, the vote of P_i is counted towards accepting the intermediary's message.

The cut and choose protocol ensures that a corrupt $\mathbf{S}$ will either be detected by the $\mathbf{M}$ forcing it to reveal the message or every honest key-holder will vote for the intermediary's message. To see this, consider any honest key-holder P_i. Consider the event in which the sender does not disqualify the sender while doing the MAC check for $k_{i,j}, j \in \mathrm{IND}_i$ and the vote of P_i is not counted towards intermediary's message. This occurs only if MAC $t_{i,j}$ sent by the corrupt sender is consistent for all $j \in \mathrm{IND}_i$ and inconsistent for all $j \notin \mathrm{IND}_i$. Since IND_i is chosen uniformly at random unknown to $\mathbf{S}$, this occurs with probability that is negligible in κ. Thus, an honest $\mathbf{M}$ will successfully call out a corrupt $\mathbf{S}$, in which case $\mathbf{S}$ reveals the message during the checking phase, or the message of $\mathbf{M}$ will be accepted by a receiver that accepts the message if a strict majority of key-holders vote for the message. Finally, a corrupt $\mathbf{M}$ that uses a value $m' \neq m$ will fail to receive a vote from any of the honest key-holders by the unforgeability of MAC, causing the receiver to reject the message.

Porting the above blueprint into a layered protocol, we encounter all the inherent challenges of any framework dealing with dynamic committees (including YOSO). In particular, the sender cannot be "present" to reveal the keys $\{k_{i,j}\}_{j \in \mathrm{IND}_i}$ after the key-holder P_i broadcasts the set IND_i. Clearly, $\mathbf{S}$ has already used up its communication round to send, among other things, the keys $\{k_{i,j}\}_{j \in [\kappa]}$ to key-holder P_i. We solve these challenges in the same way as all constructions dealing with dynamic committees: parties who need to speak multiple times cache their messages using future broadcast, and later layers conditionally reveal only those messages that are required to be opened.

However, the layered model brings up many subtle challenges that are not encountered in YOSO. It is crucial that the message and MACs are chosen by $\mathbf{S}$ *before* each honest key-holder P_i reveals their set IND_i. Otherwise, a corrupt $\mathbf{S}$ can choose the MAC tags $t_{i,j}$ to be consistent with $k_{i,j}$ for all $j \in \mathrm{IND}_i$ and inconsistent for all $j \notin \mathrm{IND}_i$. Since none of our communication primitives commit the sender to their messages, this is possible only if the layer in which the key-holders make their random sets public comes after the layer in which $\mathbf{M}$ is placed. Note, that this is not the case in YOSO: thanks to the *assumed* channels to the future, the messages from $\mathbf{S}$ to $\mathbf{M}$ are fixed at the time of sending. This allows $\mathbf{M}$, to perform the MAC checks locally after the random sets and the keys in those sets are revealed, and only speak after this check. In the layered protocol, on the other hand, the MAC checks–a non-linear operation–need to be carried out by a future layer that learns IND_i and has access to cached values of the message and MAC tags $m, t_{i,j}$ provided by $\mathbf{M}$, as well as the keys $k_{i,j}$ provided by $\mathbf{S}$. We construct this by having the sender's keys made public first, and then using them to securely compute the appropriate linear combination $m, t_{i,j}$ that yields 0 if and only if the verification succeeds. We crucially use the fact that m is not revealed in this secure computation when $\mathbf{S}$ and $\mathbf{M}$ are honest.

Finally, because the receivers perform the MAC checks using the keys provided by the key-holders, to ensure security against a dishonest $\mathbf{M}$, it is crucial that the message and MACs are revealed by $\mathbf{M}$ *before* the keys are revealed. We encountered the same challenge in future broadcast, and address it similarly: the message and MACs are made public before the keys are revealed, by adding some “dummy layers”.

2.6 Distributed Commitment

Our information theoretic signature primitive commits a dishonest sender to their input *only* when the intermediary is honest. In contrast, the distributed commitment primitive commits a sender to their input unconditionally, by leveraging the honest majority in each layer. More specifically, we realize the following functionality: a party from an initial layer can commit to one (or multiple) values towards many future layers, who can then decide whether to open (any linear combination of) the committed values. Commitments can be opened publicly (towards all parties in one layer) or privately towards a single party. The opening of commitments made by an honest party never fails, but the adversary can prevent the opening of commitments of dishonest parties.

The protocol we present follows a natural blueprint in which the committer produces a Shamir sharing of their input, and uses one instance of our information theoretic signature to sign each share; in this step, it is crucial that the intermediaries involved in the signing of each share are distinct parties in the same layer: this guarantees that at most t of them are corrupted, preserving the privacy of the committed message. Furthermore, there are at least $n - t \geq t + 1$ honest intermediaries, and the information theoretic signature primitive guarantees that their shares will be accepted. Therefore, even if the committer is corrupted, these $t + 1$ shares uniquely determine a committed value. By controlling the dishonest shares, a corrupt dealer can still open the value $\perp$, or in other words refuse to open their commitment. Details are in Section 7.

2.7 Verifiable Secret Sharing

The last ring in this chain of primitives with increasingly strong commitment guarantees is verifiable secret sharing (VSS). This can be thought of as a distributed commitment primitive in which *even a dishonest* committer cannot prevent their commitment from being opened.[2]

As none of the few VSS constructions in the setting of dynamic committees can be adapted to the layered setting with $t < n/2$, we design an entirely new protocol. Our construction makes black-box use of a linearly homomorphic (for commitments generated by the same party) distributed commitment primitive to construct a full-fledged VSS. We reduce the security of VSS to that of the underlying distributed commitment perfectly: meaning

[2]We are faced with a dichotomy of languages: a VSS protocol can be thought of as a *strong distributed commitment*. From this perspective, the party providing input is a *committer*, producing *commitments* that can be *opened*. More often, the language of *secret sharing* is used. Here, the party providing input is a *dealer*, producing *sharings* that can be *reconstructed*. We oscillate between these two abstractions.

that this construction does not introduce any further error probability. To the best of our knowledge, this simple black-box compiler is of independent interest even in the non-layered setting, where parties can send messages in multiple rounds. We provide a description of its non-layered version below. The layered version of the protocol is presented in Section 8.

An important technical point is that the resulting VSS is homomorphic even across sharings (commitments) dealt by *different* dealers, despite the fact that the underlying distributed commitment protocol is only homomorphic with respect to sharings dealt by the same dealer. This is crucial in our circuit evaluation protocol. The sharing phase works as follows.

1. The dealer uniformly samples a random bi-variate polynomial $F(x, y)$ of degree at most t in each variable conditioned on $F(0, 0) = s$, and sends to each P_i the vertical projection $F(i, y)$. The dealer also commits to (all coefficients of) the polynomial $F(x, y)$ via the distributed commitment primitive.

2. Each P_i commits to (all coefficients of) the received polynomial $v_i(y)$ via the distributed commitment primitive.

3. Using the homomorphism of the distributed commitments, parties privately open towards each party P_j the j-th horizontal projection $F(x, j)$ committed by the dealer, and the j-th evaluation point $v_i(j)$ of every P_i's committed polynomial.

4. If the private reconstructions do not match, i.e. $F(i, j) \neq v_i(j)$ (or both fail), then P_j broadcasts a complain message $(\mathsf{complain}, i, j)$.

5. For each complaint $(\mathsf{complain}, i, j)$ parties publicly open the commitments to the two points, which we denote $\overline{F}(i, j)$ from the dealer and $\overline{v}_i(j)$ from P_i. If the dealer's opening fails, disqualify the dealer. And if P_i's opening does not match the dealer's point (or fails), then add P_i to a global set $\mathcal{I}$ of parties, and publicly open the projection $F(i, y)$. The dealer is disqualified if $|\mathcal{I}| > t$.

If the dealer is not disqualified, each party $P_i \notin \mathcal{I}$ has a committed polynomial $v_i(y)$ which is the same as the vertical projection of the polynomial $F'(x, y)$ committed by the dealer in the first step. If this was not the case, the two polynomials would differ in at least $t+1$ points, and therefore one honest party P_j would have complained in Step 4, a complaint that would have led to including P_i to the set $\mathcal{I}$, a contradiction. Moreover, if the dealer is honest, it is easy to see that $F'(x, y) = F(x, y)$. Further note that the sharing phase is homomorphic across different dealers, since the distributed vertical polynomials used for reconstruction are dealt by each of the recipients, or publicly known.

To ensure reconstruction (i.e. to open the dealer' commitment), parties simply open the vertical projections committed by each $P_i \notin \mathcal{I}$, and use any $t + 1$ polynomials

$$\{v_{i_1}(y), \ldots, v_{i_{t+1}}(y)\} \tag{1}$$

that are either reconstructed in this step, or were revealed in Step 5, to interpolate the original secret.

2.8 Multi-Party Computation

The circuit evaluation protocol proceeds gate-by-gate, with gates in the same layer of the circuit being evaluated in parallel. We maintain the following invariant: there is layer of parties, say $\mathcal{L}_0$, holding a state encoding the input values a and b for each gate g in a certain level of the circuit C and a layer $\mathcal{L}_k$ holding a state encoding the output of g. The state encoding an input value a to a gate g has three components:

1. A (t, n)-Shamir sharing of a: each party $P_i^0 \in \mathcal{L}_0$ holds share $a_i = f_a(i)$.
2. A (t, n)-Shamir sharing of a random value r, which is wasted to compute multiplication gates. This can be easily achieved using our VSS protocol, by letting all parties in a layer verifiably share a random value and taking the sum.
3. Verifiable sharings (aka commitments) to the coefficients of the polynomial $f_a(x)$ used for the Shamir Sharing of a.

Input Gates. Each client needs to produce the state described above (encoding their input) towards the layer tasked with computing the first level of gates in the circuit. To achieve this, a client with input m simply commits via VSS towards two different layers to each coefficient of a degree t polynomial $f(x)$ with $f(0) = m$. Then, the intermediary layer reconstructs each $f(i)$ (exploiting the linearity of the VSS) towards party P_i in the second layer.

Addition Gates. Since the invariant state is linear, addition gates can be performed locally. However, as captured by the parallel functionality $\mathcal{F}_{\mathsf{VSS}}$ in Section 8, our VSS only allows to add sharings made by dealers in the *same layer*. The evaluation of multiplication gates, on the other hand, requires several rounds of interaction. We need the output of the addition gates to also be processed in the same number of rounds, and to avoid introducing extra machinery, we simply multiply the output of each addition gate by 1.

Multiplication Gates. The multiplication follows the classical blueprint of [18], adapted to the layered setting and the described state invariant. Suppose that layer $\mathcal{L}_0$ holds the states for the inputs a and b of the multiplication gate g. Each party P_i^0 locally multiplies their Shamir shares to compute $c_i = a_i \cdot b_i$. To reproduce the state for the value c_i, the party simply performs the client input routine described above. Note that $c = a \cdot b$ is a linear combination of the c_i's, so to compute the invariant state for c it is enough that the parties produce correct states for each c_i.

To show that the party P_i^0 actually produced a state for the right value c_i, the party provides a distributed zero-knowledge proof. Assume that the commitments to the coefficients of $f_a(x)$ and $f_b(x)$ are also available towards any auxiliary layer $\mathcal{L}_{k'}$ for $0 \leq k' \leq k$. Then, P_i^0 produces new verifiable secret sharings to $\widehat{a}_i, \widehat{b}_i$ and $\widehat{c}_i$, and proves towards parties in $\mathcal{L}_{k'}$: 1) that $\widehat{a}_i = a_i$, 2) that $\widehat{b}_i = b_i$, and 3) that $\widehat{a}_i \cdot \widehat{b}_i = \widehat{c}_i$. If the proof fails, parties in $\mathcal{L}_{k'}$ simply reveal values a_i and b_i to parties in layer $\mathcal{L}_k$.

For the proof of equality, party P_i^0 verifiably shares $\widehat{r_i}$ towards all future layers until $\mathcal{L}_k$. Then, parties in some later layer (who also hold commitments to a_i, r_i and $\widehat{a_i}$) sample a public common random value ρ, and the values $r_i + \rho a_i$ and $\widehat{r_i} + \rho \widehat{a_i}$ are opened publicly. If they are different, the proof fails. Note that if $a_i \neq \widehat{a_i}$ or $r_i \neq \widehat{r_i}$, there is only one value ρ that makes the proof succeed.

The proof of correct multiplication showing $\widehat{a_i} \cdot \widehat{b_i} = \widehat{c_i}$ is an adaptation from [12]. For this, party P_i^0 samples a random value β and verifiably shares β and $b\beta$. Now, parties in some later layer (who also hold commitments to $\widehat{a_i}, \widehat{b_i}$ and $\widehat{c_i}$) sample a random value ρ, and publicly open the value $\rho' = \rho a + \beta$. Finally, a later layer publicly opens and checks that $\rho' b - b\beta - rc = 0$. Note that if $\widehat{a_i} \cdot \widehat{b_i} \neq \widehat{c_i}$, only one value ρ makes the proof succeed, which happens with negligible probability. See details in Section 10.

3 Preliminaries

3.1 Model

A layered MPC protocol can be viewed as a special case of standard MPC with a general adversary structure, specialized in the following way: 1) the interaction pattern is defined by a layered graph, and 2) the adversary can corrupt at most t parties in each layer.

Definition 1 (Layered Protocol). *Let n, t, d be positive integers. An (n, t, d)*-layered protocol *is a synchronous protocol Π over secure point-to-point channels and a broadcast channel, with the following special features.*

- **Parties.** *There are $N = n(d+1)$ parties partitioned into $d+1$ layers $\mathcal{L}_i$, $0 \leq i \leq d$, where $|\mathcal{L}_i| = n$. Parties in the first layer $\mathcal{L}_0$ and the last layer $\mathcal{L}_d$ are referred to as* input clients *and* output clients*, respectively.*

- **Interaction pattern.** *The interaction consists of d rounds, where in round i parties in $\mathcal{L}_{i-1}$ may send messages to parties in $\mathcal{L}_i$ over secure point-to-point channels. We additionally allow each party in $\mathcal{L}_{i-1}$ to send a broadcast message to all parties in $\mathcal{L}_i$.*

- **Functionalities.** *We consider functionalities f that take inputs from input clients and deliver outputs to output clients.*

- **Adversaries.** *We consider adversaries who may corrupt any number of input and output clients, and additionally corrupt t parties in each intermediate layer $\mathcal{L}_i$, $0 < i < d$. We consider active, rushing and non-retroactive adaptive adversaries*[3]*.*

We say that a protocol Π is a layered protocol for $\mathcal{F}$ if it UC-realizes $\mathcal{F}$ in the setting of general adversary structures [7, 6, 22]*. We consider statistical security (with guaranteed output delivery) where κ denotes the security parameter and $\mathbb{F}$ is a finite field of size $1/\mathsf{negl}(\kappa)$.*

[3]For simplicity, we consider the notion of "non-retroactive" (see [11], Definition 3) adaptive adversaries, who chooses at each round r a set of up to t parties from layer $\mathcal{L}_r$ to corrupt. Since our protocols are information-theoretic, we conjecture that they are also secure against the stronger notion of retroactive-adaptive adversaries that can corrupt parties in previous layers.

3.1.1 A Note on Synchronous Universal Composability

We are interested in realizing functionalities f that take input from the input clients in layer $\mathcal{L}_0$ by default and deliver outputs to the output clients in the last layer (layer $\mathcal{L}_k$) of a layered network. We develop a synchronous protocol for computing general functionalities in the UC model. The standard UC model is asynchronous by default, but there have been a number of works that modeled synchronous universally composable frameworks [25, 7, 30, 24, 28], and our protocols can be described in any of those models. Very roughly, typically one considers a functionality that keeps track of the current round number and synchronizes the activation pattern. For example, in [25], parties have access to a clock functionality in the real world, and can query it to learn the current round number. The clock then advances only when all honest parties have queried the functionality (this ensures that honest parties remain synchronized throughout the protocol). The ideal functionality then keeps track of the activation pattern and also advances a round whenever a round-robin of activations happen. For simplicity and ease of exposition, our descriptions omit the clock and it is understood that protocols and ideal functionalities know the current round number.

3.1.2 On Ideal Broadcast Channels

Without assuming broadcast, information theoretic MPC for general functionalities is provably impossible for $t \geq n/3$ [26]. In the original layered abstraction, the ideal broadcast channels are only available between parties in immediately adjacent layers. However, while describing our constructions, we always assume that broadcast messages from a certain layer are available to parties in all later layers (but only actually need these messages up to a constant number of layers in the future). This is without loss of generality, as in the honest majority setting broadcast messages can be propagated through layers via bilateral channels and sequential majority decisions, at the price of an additional quadratic factor in communication.

3.2 Statistical Message Authentication Codes

3.2.1 Definitions

Definition 2. *A message authentication code (MAC) is a couple* $(\mathsf{Aut}, \mathsf{Vfy})$ *where* $\mathsf{Aut} : \mathcal{M} \times \mathcal{K} \to \mathcal{T}$ *and* $\mathsf{Vfy} : \mathcal{T} \times \mathcal{M} \times \mathcal{K} \to \{0,1\}$ *are efficient algorithms such that*

$$\mathsf{Vfy}(\mathsf{Aut}(m,k), m, k) = 1 \tag{2}$$

for all $m \in \mathcal{M}$ *and all* $k \in \mathcal{K}$.

Definition 3. *Let* $\mathcal{D}$ *be a distribution on* $\mathcal{K}^\ell$. *We say that a MAC scheme is* $\mathcal{D}$*-linear if* $\mathcal{M}$, $\mathcal{K}$ *and* $\mathcal{T}$ *are* $\mathbb{F}$*-vector spaces and for all* $\ell \in \mathbb{N}$, *for keys* $(k_1, \ldots, k_\ell) \leftarrow_{\mathcal{D}} \mathcal{K}^\ell$, *all messages* $(m_1, \ldots, m_\ell) \in \mathcal{M}^\ell$, *and all linear functions* $L : \mathbb{F}^\ell \to \mathbb{F}$ *it holds that*

$$\mathsf{Vfy}\big(L(k_1, \ldots, k_\ell), L(m_1, \ldots, m_\ell), L\left(\mathsf{Aut}(k_1, m_1), \ldots, \mathsf{Aut}(k_\ell, m_\ell)\right)\big) = 1. \tag{3}$$

Security. Let $\mathcal{D}$ be a distribution on $\mathcal{K}^\ell$ and let $(\mathsf{Aut}, \mathsf{Vfy})$ be a $\mathcal{D}$-linear MAC scheme. Consider the following game.

Game $\mathsf{CorrForge}_{\mathcal{D},q}^{(\mathsf{Aut},\mathsf{Vfy})}$

- Sample $(k_1, \dots, k_\ell) \leftarrow_{\mathcal{D}} \mathcal{K}$;
- $\mathcal{A}$ can make up to $q \leq \ell$ queries $(m_1, j_1), \dots, (m_q, j_q) \in \mathcal{M} \times [\ell]$ and receive the corresponding tags $t_i = \mathsf{Aut}(m_i, k_{j_i})$;
- $\mathcal{A}$ provides a triple (m', t', L) where $m' \in \mathcal{M}$, $t' \in \mathcal{T}$ and $L : \mathbb{F}^q \to \mathbb{F}$.
- The game output is 1 if and only if $\mathsf{Vfy}\big(t', m', f_L(k_{j_1}, \dots, k_{j_q})\big) = 1$ and $m' \neq L(m_1, \dots, m_q)$.

Notice that one could extend the game to allow $\mathcal{A}$ to make multiple queries for different messages under the *same* key. Since in our protocols $\mathcal{A}$ can never observe multiple MAC tags for different messages under the same key, we only present simpler constructions and we do not find it necessary to do things in full generality. Let $\mathsf{Adv}_{\mathcal{D},q}^{\mathsf{CorrForge}}(\mathcal{A})$ denote the probability that the output of the game $\mathsf{CorrForge}_{\mathcal{D},q}^{(\mathsf{Aut},\mathsf{Vfy})}(\mathcal{A})$ is 1.

Definition 4. *A $\mathcal{D}$-linear MAC* $(\mathsf{Aut}, \mathsf{Vfy})$ *is statistically $(\mathcal{D}, q, \varepsilon)$-unforgeable if for all adversaries $\mathcal{A}$ it holds that* $\mathsf{Adv}_{\mathcal{D},q}^{\mathsf{CorrForge}}(\mathcal{A}) \leq \varepsilon$.

3.2.2 Construction of a Secure MAC

Let $\mathbb{F}$ be a finite field. Let $= \mathcal{T} = \mathbb{F}$ and $\mathcal{K} = \mathbb{F}^2$. We define

$$\begin{aligned} \mathsf{Aut} : \quad \mathbb{F} \times \mathbb{F}^2 &\to \mathbb{F} \\ \big(m, (\alpha, \beta)\big) &\mapsto \alpha \cdot m + \beta. \end{aligned} \tag{4}$$

Let $A, B_1, \dots, B_\ell$ be independent uniform random variables on $\mathbb{F}$. Consider the distribution $\mathcal{D}$ of $\big(K_1, \dots, K_\ell\big)$ where $K_i = (A, B_i)$.

Lemma 1. *The MAC scheme* $(\mathsf{Aut}, \mathsf{Vfy})$ *from equation* (4) *is $\mathcal{D}$-linear: for a linear function $L : \mathbb{F}^\ell \to \mathbb{F}$ we have $f_L\big((\alpha, \beta_1), \dots, (\alpha, \beta_\ell)\big) = (\alpha, L(\beta_1, \dots, \beta_\ell))$.*

Proof. For all $\alpha, \beta_1, \dots, \beta_\ell \in \mathbb{F}$, for all $m_1, \dots, m_\ell \in \mathbb{F}$ we have

$$\begin{aligned} &\mathsf{Aut}\big((\alpha, \beta_1), m_1\big) + \dots + \mathsf{Aut}\big((\alpha, \beta_\ell), m_\ell\big) = \\ &\alpha m_1 + \beta_1 + \dots + \alpha m_\ell + \beta_\ell = \\ &\alpha(m_1 + \dots + m_\ell) + (\beta_1 + \dots + \beta_\ell) = \\ &\mathsf{Aut}\big((\alpha, \beta_1 + \dots + \beta_\ell), m_1 + \dots + m_\ell\big) = \\ &\mathsf{Aut}\Big(f_L\big((\alpha, \beta_1), \dots, (\alpha, \beta_\ell)\big), L(m_1, \dots, m_\ell)\Big). \end{aligned} \tag{5}$$

□

Lemma 2. *The MAC* $(\mathsf{Aut}, \mathsf{Vfy})$ *from equation* (4) *is statistically* $(\ell, 1/|\mathbb{F}|)$*-unforgeable under* $\mathcal{D}$*-correlated keys.*

Proof. For clarity we prove the statement for $\ell = 2$, the general case follows by induction on ℓ. Let $(k_1, k_2) \leftarrow_{\mathcal{D}} \mathbb{F}^2$. Let $m_1, m_2 \in \mathbb{F}$, and let $t_i = \mathsf{Aut}(m_i, k_{j_i}) \in \mathbb{F}$ for $i \in 1, 2$.[4] For all linear functions $L : \mathbb{F}^2 \to \mathbb{F}$, all $m', t' \in \mathbb{F}$ such that $m' \neq L(m_1, m_2)$ we have

$$\begin{aligned}
&\Pr\left[\mathsf{Vfy}\big((f_L(k_1, k_2), m', t'\big) = 1 \mid \mathsf{Aut}\,(k_1, m_1) = t_1 \wedge \mathsf{Aut}\,(k_2, m_2) = t_2\right] = \\
&\Pr\left[\alpha m' + L(\beta_1, \beta_2) = t' \mid \beta_1 = t_1 - \alpha m_1 \wedge \beta_2 = t_2 - \alpha m_2\right] \\
=&\Pr\left[\alpha m' + L(t_1 - \alpha m_1, t_2 - \alpha m_2) = t'\right] \\
=&\Pr\left[\alpha \cdot \big(m' - L(m_1, m_2)\big) + L(t_1, t_2) = t'\right] \\
=&\Pr\left[\alpha = \frac{t' - L(t_1, t_2)}{m' - L(m_1, m_2)}\right] = \frac{1}{|\mathbb{F}|}.
\end{aligned} \tag{6}$$

This shows that for all adversaries $\mathcal{A}$ and all possible transcripts $\mathbf{t}$ the probability that an adversary wins the game given a certain transcript is at most $1/|\mathbb{F}|$. Therefore we have

$$\begin{aligned}
&\Pr\left[\mathsf{CorrForge}_{\mathcal{D},\ell}^{(\mathsf{Aut},\mathsf{Vfy})}(\mathcal{A}) = 1\right] \\
=&\sum_{\mathbf{t}} \Pr\left[\mathsf{CorrForge}_{\mathcal{D},\ell}^{(\mathsf{Aut},\mathsf{Vfy})}(\mathcal{A}) = 1 \mid \mathbf{t}\right] \cdot \Pr\left[\mathbf{t}\right] \\
\leq&\frac{1}{|\mathbb{F}|} \sum_{\mathbf{t}} \Pr[\mathbf{t}] = \frac{1}{|\mathbb{F}|}
\end{aligned} \tag{7}$$

□

3.3 Statistical Robust Secret Sharing Schemes

3.3.1 Definitions

Definition 5. *(Linear Secret Sharing) A* $\mathcal{D}$*-linear* (t, n)*-secret sharing scheme over a* $\mathbb{F}$*-vector space* $\mathcal{M}$ *is a tuple* $(\mathsf{Sh}, \mathsf{Rec})$ *of algorithms such that*

- $\mathsf{Sh}(s, r) \to (s_1, \ldots, s_n)$ *that takes as input a message* $s \in \mathcal{M}$ *and randomness* $r \in \mathcal{R}$ *and outputs shares* $(s_1, \ldots, s_n) \in \mathcal{S}^n$ *where the* $\mathbb{F}$*-vector space* $\mathcal{S}$ *is referred to as the share space.*
- $\mathsf{Rec}(s_1, \ldots, s_n) \to s'$ *takes as input an element* $(s_{i_1}, \ldots, s_{i_{t+1}}) \in \mathcal{S}^{t+1}$ *and outputs a message* $s' \in \mathcal{M}$.
- ***(Privacy)*** *For all random variables* S *on* $\mathcal{S}$ *and uniform random variable* R *on* $\mathcal{R}$*, and for all subsets* $\mathcal{I} \subseteq [n]$ *such that* $|\mathcal{I}| \leq t$ *we have*

$$S \sim \Big(S \mid \big(\mathsf{Sh}(S, R)_i\big)_{i \in \mathcal{I}}\Big). \tag{8}$$

[4] We can assume without loss of generality that $j_i = i$ since the distribution $\mathcal{D}$ is symmetric.

- *(**Correctness and $\mathcal{D}$-Linearity**) Assume that $\mathcal{M}$ and $\mathcal{S}$ are $\mathbb{F}$-vector spaces and let $\mathcal{D}$ be a distribution on $\mathcal{R}^\ell$. We require that for all $s_1, \ldots, s_\ell \in \mathcal{M}$, $r_1 \ldots, r_\ell \leftarrow_D \mathcal{R}^\ell$, and all linear functions $L : \mathbb{F}^\ell \to \mathbb{F}$ we have*

$$\begin{aligned}\mathsf{Rec}\Big(&L\big(\mathsf{Sh}(s_1, r_1)_{i_1}, \ldots, \mathsf{Sh}(s_\ell, r_\ell)_{i_1}\big),\\ &\ldots,\\ &L\big(\mathsf{Sh}(s_1, r_1)_{i_{t+1}}, \ldots, \mathsf{Sh}(s_\ell, r_\ell)_{i_{t+1}}\big)\Big) =\\ &L(s_1, \ldots, s_\ell).\end{aligned} \tag{9}$$

Notice that when L is simply the projection onto the k-th component, if the support of each marginal distribution of $\mathcal{D}$ is the entire set $\mathcal{R}$ then we get the typical correctness property of secret sharing, that is for all $s \in \mathcal{S}$ and $r \in \mathcal{R}$

$$\mathsf{Rec}\Big(\mathsf{Sh}(s, r)_{i_1}, \ldots, \mathsf{Sh}(s, r)_{i_{t+1}}\Big) = s. \tag{10}$$

We want to define a notion of *robustness* for secret sharing, ensuring that even when an adversary can maliciously modify some of the shares of a well-formed sharing, the reconstruction procedure still outputs the intended value. This should still hold when an adversary is given access to a subset of shares of multiple secrets computed under correlated keys. Let $\mathcal{D}$ be a distribution on $\mathcal{R}^\ell$ for some $\ell \in \mathbb{N}$. Let $(\mathsf{Sh}, \mathsf{Rec})$ be a $\mathcal{D}$-linear (t, n)-secret sharing scheme with secret space $\mathcal{M}$ and randomness space $\mathcal{R}$. Consider the following security game, where $q \leq \ell$.

Game $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{Sh},\mathsf{Rec})}(\mathcal{A})$

1. $\mathcal{A}$ picks secrets $s^{(1)}, \ldots, s^{(q)} \in \mathcal{M}$.
2. Sample $r_1, \ldots, r_q \leftarrow_{\mathcal{D}} \mathcal{R}$, compute $(s_1^{(i)}, \ldots, s_n^{(i)}) = \mathsf{Sh}(s^{(i)}, r_i)$ for $i \in [q]$.
3. $\mathcal{A}$ picks an index $i \in [n]$ and is given shares $\left(s_i^{(1)}, \ldots, s_i^{(q)}\right)$. Repeat this step at most t times. Let $\mathcal{I}$ denote the set of indices queried by $\mathcal{A}$.
4. $\mathcal{A}$ picks a vector $(s'_i)_{i \in \mathcal{I}}$ and a linear function $L : \mathbb{F}^q \to \mathbb{F}$.
5. Let
$$\hat{s}_i = \begin{cases} L\Big(s_i^1, \ldots, s_i^{(q)}\Big) \text{ if } i \notin \mathcal{I}, \\ s'_i \text{ if } i \in \mathcal{I}. \end{cases} \tag{11}$$
Return 1 if and only if $\mathsf{Rec}(\hat{s}_1, \ldots, \hat{s}_n) \neq L\big(s^{(1)}, \ldots, s^{(q)}\big)$.

Definition 6 (Robust Secret Sharing)**.** *We say a $\mathcal{D}$-linear (t, n)-secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ is $(\mathcal{D}, q, \varepsilon)$-robust if for all adversaries $\mathcal{A}$ it holds that*

$$\Pr[\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{Sh},\mathsf{Rec})}(\mathcal{A}) = 1] \leq \varepsilon. \tag{12}$$

Observe that this security game only captures attacks where the adversary modifies corrupted shares *independently* from the remaining honest shares. By allowing interactive reconstruction procedures, even in the $t < n/2$ setting it is possible to achieve a stronger security notion, tolerating attacks in which the corrupt shares are chosen after having seen *all* the honest shares. Indeed, our future broadcast protocol from Section 5 can be shown to satisfy this stronger guarantee. However, since we capture the security of future broadcast via a stronger simulation-based definition, we do not treat such a security notion separately here.

3.3.2 Construction of a Robust Secret Sharing Scheme

Consider a $\mathcal{D}'$-linear (q, ε)-secure MAC system $(\mathsf{Aut}, \mathsf{Vfy})$ with message space $\mathcal{M}$, key space $\mathcal{K}$ and tag space $\mathcal{T}$, as well as a $\mathcal{U}$-linear[5] (t, n)-secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ with secret space $\mathcal{M}$, randomness space $\mathcal{R}$ and share space $\mathcal{S}$. Consider the following couple of efficient algorithms $(\mathsf{AuthShare}, \mathsf{AuthRec})$. Let $s \in \mathcal{M}$ and $r = \left(K, r'\right) \in \mathcal{K}^{n^2} \times \mathcal{R}$.

Robust Secret Sharing

Sharing Algorithm $\mathsf{AuthShare}(s, r)$

Given message and randomness (s, r) as inputs do:

–
$$\mathbf{K} = \begin{pmatrix} k_{11} & \dots & k_{1n} \\ \vdots & \ddots & \vdots \\ k_{n1} & \dots & k_{nn}. \end{pmatrix}. \tag{13}$$

– $(s'_1, \dots, s'_n) \leftarrow \mathsf{Sh}(s, r')$.

– Compute the following MAC tags:

$$\mathbf{T} = \begin{pmatrix} t_{11} & \dots & t_{1n} \\ \vdots & \ddots & \vdots \\ t_{n1} & \dots & t_{nn} \end{pmatrix} = \begin{pmatrix} \mathsf{Aut}\left(k_{11}, s'_1\right) & \dots & \mathsf{Aut}\left(k_{1n}, s'_n\right) \\ \vdots & \ddots & \vdots \\ \mathsf{Aut}\left(k_{n1}, s'_1\right) & \dots & \mathsf{Aut}\left(k_{nn}, s'_n\right) \end{pmatrix} \tag{14}$$

– $s_i \leftarrow (s'_i, (k_{11}, \dots, k_{1n}), (t_{11}, \dots, t_{n1}))$.

– Output $(s_1, \dots, s_n)$.

Reconstruction Algorithm $\mathsf{AuthRec}(s, r)$

Given shares $(\hat{s}_1, \dots, \hat{s}_n)$ as input do:

[5] By $\mathcal{U}$ we denote the uniform random distribution on $\mathcal{R}^{\ell}$.

– *Accept* a share $\hat{s}_i$ if and only if

$$|\{j \in [n] \mid \mathsf{Vfy}(t_{ij}, s_i, k_{ij}) = 1\}| \geq t + 1. \tag{15}$$

– Let $\mathcal{G}$ denote the set of indices of the first $t+1$ accepted shares. Output $s' = \mathsf{Rec}((\hat{s}_i)_{i \in \mathcal{G}})$.

Lemma 3. *Let $\mathcal{D}$ be the distribution $(\mathcal{D}', \mathcal{U})$ on $\mathcal{R}' = \mathcal{K}^{n^2} \times \mathcal{R}$. Then $(\mathsf{AuthShare}, \mathsf{AuthRec})$ is a $\mathcal{D}$-linear (t,n)-secret sharing scheme with message space $\mathcal{M}' = \mathcal{M}$, randomness space $\mathcal{R}'$ and secret space $\mathcal{S}' = \mathcal{S} \times \mathcal{K}^n \times \mathcal{T}^n$.*

Proof. $\mathcal{D}$-linearity follows from direct computation applying the $\mathcal{D}'$-linearity of $(\mathsf{Aut}, \mathsf{Vfy})$ and $\mathcal{U}$-linearity of $(\mathsf{Sh}, \mathsf{Rec})$. Similarly, privacy and correctness follow from the corresponding properties of the underlying schemes. □

Lemma 4. *If $(\mathsf{Sh}, \mathsf{Rec})$ is a $\mathcal{U}$-linear (t,n)-secret sharing scheme and $(\mathsf{Aut}, \mathsf{Vfy})$ is a $(\mathcal{D}', q, \varepsilon)$-secure MAC scheme, then $(\mathsf{AuthShare}, \mathsf{AuthRec})$ is a $(\mathcal{D}, q, \delta)$-robust secret sharing scheme where $\mathcal{D} = (\mathcal{D}', \mathcal{U})$, that is*

$$\Pr[\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{AuthShare},\mathsf{AuthRec})}(\mathcal{A}) = 1] \leq \varepsilon \cdot tn. \tag{16}$$

Proof. First, we provide some intuition. Let $\mathcal{I}$ denote the set of shares corrupted by $\mathcal{A}$ in $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{AuthShare},\mathsf{AuthRec})}(\mathcal{A})$. Consider the event that for some $i \in \mathcal{I}$ a wrong share $\hat{s}_i \neq L(s^{(1)}, \ldots, s^{(q)})$ is *accepted* in $\mathsf{AuthRec}$. Clearly, this event can only occur if at least $t+1$ of the MAC tags t_{ij} of s'_i (recall that s'_i is composed of a proper *share* s'_i as well as n of MAC tags and n MAC keys) pass verification under the corresponding linear combination L of keys contained in shares $s_j^{(1)}, \ldots, s_j^{(q)}$. Since $|\mathcal{I}| \leq t$ then at least for one j the corresponding linear combinations of keys is not known to the adversary. This should violate security of the MAC scheme. We formalize this argument by providing an appropriate reduction: given an adversary $\mathcal{A}$ such that $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{AuthShare},\mathsf{AuthRec})}(\mathcal{A}) \geq \varepsilon$ we produce an adversary $\mathcal{A}'(\mathcal{A})$ against $\mathsf{CorrForge}_{\mathcal{D}',q}^{(\mathsf{Aut},\mathsf{Vfy})}(\mathcal{A}') \geq \frac{\varepsilon}{nt}$. The adversary $\mathcal{A}'$ needs to simulate the game $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{AuthShare},\mathsf{AuthRec})}$ for $\mathcal{A}$. Upon receiving queries $s^{(1)}, \ldots, s^{(q)}$ from $\mathcal{A}$, for all $j \in [q]$ adversary $\mathcal{A}'$ samples $r^{(j)} \leftarrow_{\mathcal{U}} \mathcal{R}$ uniformly at random and computes $\left(s_1'^{(j)}, \ldots, s_n'^{(j)}\right) \leftarrow \mathsf{Sh}\left(s^{(j)}, r^{(j)}\right)$ for all $j \in [q]$. Then, adversary $\mathcal{A}'$ queries $\mathsf{CorrForge}_{\mathcal{D}',q}^{(\mathsf{Aut},\mathsf{Vfy})}$ with $s_1'^{(1)}, \ldots, s_1'^{(q)}$ receiving MAC tags $t^{(1)}, \ldots, t^{(q)}$. After this, for all $i, j \in [n]$ the adversary $\mathcal{A}'$ samples keys $\left(k_{i,j}^{(1)}, \ldots, k_{i,j}^{(q)}\right) \leftarrow_{\mathcal{D}'} \mathcal{K}$ and computes[6]

$$T^{(i)} = \begin{pmatrix} t^{(i)} & \ldots & \mathsf{Aut}\left(k_{1,n}^{(i)}, s_n^{(i)}\right) \\ \vdots & \ddots & \vdots \\ \mathsf{Aut}\left(k_{n,1}^{(1)}, s^{(1)}\right) & \ldots & \mathsf{Aut}\left(k_{n,n}^{(i)}, s_n^{(i)}\right) \end{pmatrix}. \tag{17}$$

[6]The position of tag $t^{(i)}$ in the matrix must actually be chosen uniformly at random (the same for all $i \in [q]$). However, to keep notation simple, we put in position $(1,1)$ of the matrix but assume its position to be uniformly random in later analysis.

Now, adversary $\mathcal{A}'$ computes $s_j^{(i)} \leftarrow \left(s_j'^{(i)}, \left(k_{j,1}^{(i)}, \ldots, k_{j,n}^{(i)}\right), \left(t_{1,j}^{(i)}, \ldots, t_{n,j}^{(i)}\right)\right)$ for all $j \neq 1$ and all $i \in [q]$ and then receives up to t indices as queries from $\mathcal{A}$. Upon being queried index $i \in [n]$ from $\mathcal{A}$, adversary $\mathcal{A}'$ gives shares $\left(s_i^{(1)}, \ldots, s_1^{(q)}\right)$ to $\mathcal{A}$, and continues answering the queries. If $\mathcal{A}$ queries index $j = 1$, then $\mathcal{A}'$ answers with fixed messages all remaining queries of $\mathcal{A}$ and sets the output of $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{AuthShare},\mathsf{AuthRec})}(\mathcal{A})$ to 0. Otherwise let $\mathcal{I} \subseteq [n]$ denote the set of indices queried by $\mathcal{A}$ with $1 \notin \mathcal{I}$. Now adversary $\mathcal{A}'$ receives the last query $\left(L, (\overline{s}_i)_{i\in\mathcal{I}}\right)$ from $\mathcal{A}$. Let $\ell \leftarrow_{\$} \mathcal{I}$. Recall the description of each corrupt share $\overline{s_i} = \left(\overline{s_i'}, \left(\overline{k_{i,1}}, \ldots, \overline{k_{i,n}}\right), \left(\overline{t_{1,i}}, \ldots, \overline{t_{n,i}}\right)\right)$. Adversary $\mathcal{A}'$ makes the forgery query $\left(L, \overline{s_i'}\overline{t_{1,i}}\right)$ to $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{AuthShare},\mathsf{AuthRec})}$. The output of $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{AuthShare},\mathsf{AuthRec})}(\mathcal{A})$ is taken to be the output of $\mathsf{CorrForge}_{\mathcal{D}',q}^{(\mathsf{Aut},\mathsf{Vfy})}(\mathcal{A}')$. Let bad denote the event that $1 \in \mathcal{I}$. We have

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{CorrForge}_{\mathcal{D}',q}^{(\mathsf{Aut},\mathsf{Vfy})}}(\mathcal{A}') = \\
&\Pr\left[\mathsf{CorrForge}_{\mathcal{D}',q}^{(\mathsf{Aut},\mathsf{Vfy})}(\mathcal{A}') = 1\right] = \\
&\Pr\left[\mathsf{CorrForge}_{\mathcal{D}',q}^{(\mathsf{Aut},\mathsf{Vfy})}(\mathcal{A}') = 1 \mid \mathsf{bad}\right] \Pr[\mathsf{bad}] \\
+&\Pr\left[\mathsf{CorrForge}_{\mathcal{D}',q}^{(\mathsf{Aut},\mathsf{Vfy})}(\mathcal{A}') = 1 \mid \neg\mathsf{bad}\right] \Pr[\neg\mathsf{bad}] = \\
&\Pr\left[\mathsf{CorrForge}_{\mathcal{D}',q}^{(\mathsf{Aut},\mathsf{Vfy})}(\mathcal{A}') = 1 \mid \neg\mathsf{bad}\right] \Pr[\neg\mathsf{bad}] = \\
&\Pr\left[\mathsf{CorrForge}_{\mathcal{D}',q}^{(\mathsf{Aut},\mathsf{Vfy})}(\mathcal{A}') = 1 \mid \mathsf{bad}\right] \cdot \frac{(n-t)}{n} = \\
&\Pr\left[\mathsf{Vfy}\left(L\left(k^{(1)}, \ldots, k^{(q)}\right), \overline{s_i'}, \overline{t_{1,i}}\right) = 1 \mid \neg\mathsf{bad}\right] \cdot \frac{(n-t)}{n}.
\end{aligned} \tag{18}$$

Now, observe that if $\neg\mathsf{bad}$ occurs, then the view of $\mathcal{A}$ in the simulated game is the same as in a real instance of $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{AuthShare},\mathsf{AuthRec})}(\mathcal{A})$. If $\mathcal{A}$ wins a real instance of the game, then it has forged the MAC tags in the corrupted shares (this is clear by inspection $\mathsf{AuthRec}$ function). Let $(\widehat{s}_i)_{i\in\mathcal{I}}$ denote the corrupted shares. A simple avaraging argument shows that for all $j \notin \mathcal{I}$ and all $i \in \mathcal{I}$

$$\begin{aligned}
&\Pr\left[\mathsf{Vfy}\left(k_{j,i}, s_i', t_{i,j}\right) = 1 \mid \mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{AuthShare},\mathsf{AuthRec})}(\mathcal{A}) = 1\right] \\
&\cdot \Pr[\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{AuthShare},\mathsf{AuthRec})}(\mathcal{A}) = 1] \geq \\
&\frac{\varepsilon}{t(n-t)},
\end{aligned} \tag{19}$$

which combined with equation (18) yields

$$\mathsf{Adv}^{\mathsf{CorrForge}_{\mathcal{D}',q}^{(\mathsf{Aut},\mathsf{Vfy})}}(\mathcal{A}') \geq \frac{\varepsilon}{t(n-t)} \cdot \frac{(n-t)}{n} = \frac{\varepsilon}{tn}. \tag{20}$$

□

Consider a $(\mathcal{D}, q, \delta)$-robust secret sharing scheme $(\mathsf{AuthShare}, \mathsf{AuthRec})$. Such a scheme allows for *efficient patching* if for $(s_1, \ldots, s_n) \leftarrow \mathsf{AuthShare}(s, r)$ where $s \in \mathcal{M}$ and $r \in \mathcal{D}$ and any subset $\mathcal{I}$ with $|\mathcal{I}| \leq t$, there exists a polynomial time computable function G such that for any $s' \in \mathcal{M}$ where $s \neq s'$, it holds that

$$\mathsf{AuthRec}\big(G(s, r, \mathcal{I}, s'), (s_j)_{j \in \mathcal{I}}\big) = s'. \tag{21}$$

That is, it is possible to explain the shares of any subset of size $|\mathcal{I}| \leq t$ as the shares of any secret $s' \in \mathcal{M}$.

Lemma 5. $(\mathsf{AuthShare}, \mathsf{AuthRec})$ *is amenable to efficient patching.*

Proof. Consider the shares $(s_1, \ldots, s_n) \leftarrow \mathsf{AuthShare}(s, r)$ where $s \in \mathcal{M}$ and $r \in \mathcal{D}$. We present a proof by construction of the patching function G. On input $(s, r, \mathcal{I}, s')$, G first obtains the aforementioned shares $(s_1, \ldots, s_n)$ where $s_i = (\hat{s}_i, (k_{i,1}, \ldots, k_{i,n}), (t_{1,i}, \ldots, t_{n,i}))$. We may assume that the underlying secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ is amenable to patching. That is, there exists an efficient function that on input $(s, r, \mathcal{I}, s')$, outputs a value r' with the following properties: (1) r' is uniformly random in $\mathcal{D}$ given s' and (2) (by construction) is the value where $(\hat{s}'_1, \ldots, \hat{s}'_n) \leftarrow \mathsf{Sh}(s', r')$ such that $\hat{s}'_i = \hat{s}_i$ for all $i \in \mathcal{I}$. What remains is to recompute tags $t_{1,i}, \ldots, t_{n,i}$ for each share s_i where $i \notin \mathcal{I}$ such that $t'_{j,i} \leftarrow \mathsf{Aut}(k_{j,i}, \hat{s}'_i)$. Finally, G outputs the final patched shares $(s'_i)_{i \notin \mathcal{I}}$ where

$$s'_i \leftarrow \left(\hat{s}'_i, (k_{i,1}, \ldots, k_{i,n}), \left(t'_{1,i}, \ldots, t'_{n,i}\right)\right). \tag{22}$$

By inspection of the above construction, G runs in polynomial time and for any $\mathcal{D}$-linear combination of shares the same constructive approach is applicable. $\square$

4 Future Messaging

4.1 Future Messaging Functionality

As discussed in the technical overview, a basic challenge in the layered setting is for a party in a layer $\mathcal{L}_0$ to communicate securely with parties in a later layer $\mathcal{L}_k$ for $k > 0$. If $k = 1$, communication happens via provided point-to-point secure channels. However, if $k \geq 2$ secure channels must be emulated via an appropriate layered protocol. Our parallel future messaging functionality allows each party in layer $\mathcal{L}_0$ to send a message to each party in a layer $\mathcal{L}_k$ for any $k \geq 1$. We remark that the guarantees provided by the functionality are quite weak, as the adversary is allowed to fix the messages from corrupted parties in $\mathcal{L}_0$ to honest parties in $\mathcal{L}_k$ *after* having received the messages sent by honest parties in $\mathcal{L}_0$ to corrupted parties in $\mathcal{L}_k$.

The parallel functionality we describe below is the strongest functionality that can be obtained by composing our protocol for a single sender and receiver in parallel, due to the rushing attacks we just described.

Parallel Future Messaging Functionality $\mathcal{F}^k_{\mathsf{FutureMsg}}$

Public Parameters. Senders $\mathbf{S}_1, \ldots, \mathbf{S}_n \in \mathcal{L}_0$, receivers $\mathbf{R}_1, \ldots, \mathbf{R}_n \in \mathcal{L}_k$ where $k \geq 1$. The domain $M_{i,j}$ of message from $\mathbf{S}_i$ to $\mathbf{R}_j$.

Secret Inputs. For each $\mathbf{S}_i$ messages $m_{i,j} \in M_{i,j}$ for $j \in [n]$ to be sent to each $\mathbf{R}_j$.

Layer $\mathcal{L}_0$:

- For each honest $\mathbf{S}_i \in \mathcal{L}_0 \setminus \mathcal{I}_0$ and each $\mathbf{R}_j \in \mathcal{L}_k$, receives message $m_{i,j}$ from $\mathbf{S}_i$ to $\mathbf{R}_j$.

Layer $\mathcal{L}_k$:

- For each honest $\mathbf{S}_i \in \mathcal{L}_0 \setminus \mathcal{I}_0$ and corrupt $\mathbf{R}_k \in \mathcal{I}_k$, forward $m_{i,j}$ to the (ideal) adversary.
- For each corrupt $\mathbf{S}_i \in \mathcal{I}_0$ and each $\mathbf{R}_j \in \mathcal{L}_k$, receive from the (ideal) adversary the message $m_{i,j}$ that $\mathbf{S}_i$ wants to send to $\mathbf{R}_j$.
- For each $\mathbf{S}_i \in \mathcal{L}_0$ and $\mathbf{R}_j \in \mathcal{L}_k$, send $m_{i,j}$ to $\mathbf{R}_j$ as message from $\mathbf{S}_i$.

4.2 Future Messaging Protocol

We give a formal description of the protocol outlined in Section 2.3 realizing functionality $\mathcal{F}^k_{\mathsf{FutureMsg}}$. The security of the protocol, captured by Lemma 6, is proven in Section 4.3. The protocol crucially relies on a $(\mathcal{D}, t, \mathsf{negl}(\kappa))$-robust (t, n)-secret sharing scheme. One such scheme is described in Section 3.3.

Parallel Future Messaging Protocol $\Pi^k_{\mathsf{FutureMsg}}$

Public Parameters. Senders $\mathbf{S}_1, \ldots, \mathbf{S}_n \in \mathcal{L}_0$. Receivers $\mathbf{R}_1, \ldots, \mathbf{R}_n \in \mathcal{L}_k$. The domain $M_{i,j}$ of message from $\mathbf{S}_i$ to $\mathbf{R}_j$.

Secret Inputs. From $\mathbf{S}_i$ a message $m_{i,j} \in M_{i,j}$ to be sent to $\mathbf{R}_j$ for $i, j \in [n]$.

Resources. A $(\mathcal{D}, t, \mathsf{negl}(\kappa))$-robust (t, n)-secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$; functionalities $\mathcal{F}^{k'}_{\mathsf{FutureMsg}}$ and $\mathcal{F}^{k-k'}_{\mathsf{FutureMsg}}$ for some $k' \in [k-1]$.

Layer $\mathcal{L}_0$: Each $\mathbf{S}_i \in \mathcal{L}_0$ samples $r_{i,j} \leftarrow_\$ \mathcal{R}$, computes

$$(m^1_{i,j}, \ldots, m^n_{i,j}) \leftarrow \mathsf{Sh}(m_{i,j}, r_{i,j}) \tag{23}$$

for all $j \in [n]$, and sets $(m^\ell_{i,1}, \ldots, m^\ell_{i,n})$ as the message [a] to $P^{k'}_\ell$ in functionality $\mathcal{F}^{k'}_{\mathsf{FutureMsg}}$.

Layer $\mathcal{L}_{k'}$: Each $P_\ell^{k'} \in \mathcal{L}_{k'}$ receives (from $\mathcal{F}^{k'}_{\mathsf{FutureMsg}}$) values $\widehat{m^\ell_{i,j}}$ for all $i \in [n]$ and $j \in [n]$, and sets the message $m_{\ell,j}$ to $\mathbf{R}_j$ in $\mathcal{F}^{k-k'}_{\mathsf{FutureMsg}}$ to be the vector

$$m_{\ell,j} = (\widehat{m^\ell_{1,j}}, \dots, \widehat{m^\ell_{1,j}}). \tag{24}$$

Layer $\mathcal{L}_k$: Each $\mathbf{R}_j$ computes their output

$$\overline{m_{i,j}} \leftarrow \mathsf{Rec}\left(\overline{m^1_{i,j}}, \dots, \overline{m^n_{i,j}}\right) \tag{25}$$

for all $i \in [n]$, where $\left(\overline{m^\ell_{1,j}}, \dots, \overline{m^\ell_{n,j}}\right)$ is the message from $P_\ell^{k'}$ to $\mathbf{R}_j$ in $\mathcal{F}^{k-k'}_{\mathsf{FutureMsg}}$.

[a] The set $M_{i,\ell}$ in $\mathcal{F}^{k'}_{\mathsf{FutureMsg}}$ is set large enough to accommodate $(m^\ell_{i,1}, \dots, m^\ell_{i,n})$.

4.3 Future Messaging Security

Lemma 6. *If* $(\mathsf{Sh}, \mathsf{Rec})$ *is a* $(\mathcal{D}, t, \mathsf{negl}(\kappa))$*-robust* (t,n)*-secret-sharing scheme, then for any* $k' \in [k-1]$ *the* (n,t,k)*-layered protocol* $\Pi^k_{\mathsf{FutureMsg}}$ *realizes functionality* $\mathcal{F}^k_{\mathsf{FutureMsg}}$ *with* $(\mathsf{negl}(\kappa), t)$*-statistical security in the* $\left(\mathcal{F}^{k'}_{\mathsf{FutureMsg}}, \mathcal{F}^{k-k'}_{\mathsf{FutureMsg}}\right)$*-hybrid model.*

Proof. For any adversary $\mathcal{A}$ we describe a simulator σ such that the joint distribution of the outputs of the adversary and honest parties in the real (hybrid) world (the adversary $\mathcal{A}$ interacting with protocol $\Pi^k_{\mathsf{FutureMsg}}$ and functionalities $\mathcal{F}^{k'}_{\mathsf{FutureMsg}}, \mathcal{F}^{k-k'}_{\mathsf{FutureMsg}}$) is statistically close to that in the ideal world (the simulator σ interacting with functionality $\mathcal{F}^k_{\mathsf{FutureMsg}}$). We will show that if this statistical distance is non-negligible, then we can produce an adversary $\mathcal{A}'(\mathcal{A})$ that wins game $\mathsf{Rob}^{(\mathsf{Sh},\mathsf{Rec})}_{\mathcal{D},q}$ with non-negligible probability, therefore reducing the security of $\Pi^k_{\mathsf{FutureMsg}}$ in the $\left(\mathcal{F}^{k'}_{\mathsf{FutureMsg}}, \mathcal{F}^{k-k'}_{\mathsf{FutureMsg}}\right)$-hybrid model to the $(\mathcal{D}, \ell, \delta)$-robustness of $(\mathsf{Sh}, \mathsf{Rec})$. First, given any adversary $\mathcal{A}$ interacting with protocol $\Pi^k_{\mathsf{FutureMsg}}$ and functionalities $\mathcal{F}^{k'}_{\mathsf{FutureMsg}}, \mathcal{F}^{k-k'}_{\mathsf{FutureMsg}}$ we describe the simulator σ.

Simulator $\sigma(\mathcal{A})$ for protocol $\Pi^k_{\mathsf{FutureMsg}}$

Layer $\mathcal{L}_0$:

- For each honest $\mathbf{S}_i \notin \mathcal{I}_0$ sample $r_{i,j} \leftarrow_\$ \mathcal{R}$ for $j \in [n]$;
- For each honest $\mathbf{S}_i \notin \mathcal{I}_0$ compute

$$\left(\widehat{m^1_{i,j}}, \dots, \widehat{m^n_{i,j}}\right) \leftarrow \mathsf{Sh}(0, r_{i,j}) \tag{26}$$

  for $j \in [n]$.[a]

Layer $\mathcal{L}_{k'}$:

- For each corrupt $P_\ell^{k'} \in \mathcal{I}_{k'}$ send to $\mathcal{A}$ values $\widehat{m_{i,j}^\ell}$ for all honest $\mathbf{S}_i \notin \mathcal{I}_0$ and $j \in [n]$ (on behalf of $\mathcal{F}_{\mathsf{FutureMsg}}^{k'}$).

- For each corrupt $\mathbf{S}_i \in \mathcal{I}_0$ receive from $\mathcal{A}$ values $\left(\widehat{m_{i,j}^1}, \ldots, \widehat{m_{i,j}^n}\right)$ for all $\ell \in [n]$.

Layer $\mathcal{L}_k$:

- For each $\mathbf{R}_j \in \mathcal{I}_k$ receive values $m_{i,j}$ for each honest $\mathbf{S}_i \in \mathcal{I}_0$ as outputs from $\mathcal{F}_{\mathsf{FutureMsg}}^{k}$.

- Patch the simulated honest states

$$\left(\widehat{m_{i,j}^1}, \ldots, \widehat{m_{i,j}^n}\right) \mapsto \left(\left(\widehat{m_{i,j}^\ell}\right)_{\ell \in \mathcal{I}_{k'}}, \left(m_{i,j}^\ell\right)_{\ell \notin \mathcal{I}_{k'}}\right) \tag{27}$$

  to match these values, as explained in Lemma 5.

- For all honest $P_\ell^{k'} \notin \mathcal{I}_{k'}$ and all corrupt $\mathbf{R}_j \in \mathcal{I}_k$ send the patched $m_{i,j}^\ell$ for all $i \in [n]$ to $\mathcal{A}$ as (on behalf of $\mathcal{F}_{\mathsf{FutureMsg}}^{k'-k}$).

- For each corrupt $P_\ell^{k'} \in \mathcal{I}_{k'}$ receive from $\mathcal{A}$ values $\overline{m_{i,j}^\ell}$ for all $i, j \in [n]$.

- For all corrupt $\mathbf{S}_i \in \mathcal{I}_0$ and honest $\mathbf{R}_j \notin \mathcal{I}_k$ compute

$$\overline{m_{i,j}} \leftarrow \mathsf{Rec}\left(\widehat{m_{i,j}^1}, \ldots, \widehat{m_{i,j}^n}\right) \tag{28}$$

  and input them to $\mathcal{F}_{\mathsf{FutureMsg}}^{k}$.

- Set output to the output of $\mathcal{A}$.

[a]Notice that 0 is en element of the $\mathbb{F}$-vector space $M_{i,j}$ for all $i, j \in [n]$.

Let's start by arguing about the view of the adversary. By inspection of the protocol and the simulation we conclude that the only difference between the view of $\mathcal{A}$ in the real world and the ideal world is that values $\widehat{m_{i,j}^\ell}$ for all $\mathbf{S}_i \notin \mathcal{I}_0$ and $j \in [n]$ received from $\mathcal{A}$ in layer $\mathcal{L}_{k'}$ in the simulation are sampled according to $\mathsf{Sh}(0, r_{i,j})$, while in the real protocol execution they are sampled according to $\mathsf{Sh}(m_{i,j}, r_{i,j})$ where $m_{i,j}$ are the real honest parties inputs. However, since $|\mathcal{I}_{k'}| \le t$, from t-privacy of $(\mathsf{Sh}, \mathsf{Rec})$ we know that any set of up to t shares is identically distributed regardless of the secret s, from which we conclude that the view of the adversary in both scenarios is identically distributed. Next, we argue about the outputs of honest parties. In the ideal world the outputs of each honest $\mathbf{R}_j \notin \mathcal{I}_k$ is simply the vector $(m_{1,j}, \ldots, m_{n,j})$ received from $\mathcal{F}_{\mathsf{FutureMsg}}^{k}$, where if $\mathbf{S}_i$ is honest then $m_{i,j}$ is the real input of $\mathbf{S}_i$, while if $\mathbf{S}_i$ is corrupt then $m_{i,j} = \mathsf{Rec}\left(\overline{m_{i,j}^1}, \ldots, \overline{m_{i,j}^n}\right)$ as set by σ. In the real world, an honest $\mathbf{R}_j$ always outputs $\overline{m_{i,j}} = \mathsf{Rec}\left(\overline{m_{i,j}^1}, \ldots, \overline{m_{i,j}^n}\right)$. If $\mathbf{S}_i$ is corrupted, then the shares are the same in the real and ideal world, so that the output of $\mathbf{R}_j$ is trivially the same. If $\mathbf{S}_i$ is honest, in the real world at least $n-t$ of the shares $\overline{m_{i,j}^\ell}$ received by $\mathbf{R}_j$ from parties $P_\ell^{k'}$ are by from $\mathbf{S}_i$ as $\mathsf{Sh}(m_{i,j}, r)$ for some $r \leftarrow_\$ \mathcal{R}$. Here, if the output $m_{i,j}$ of $\mathbf{R}_j \notin \mathcal{I}_k$ is different in the real and ideal world, then $\mathcal{A}$ can be used to construct an adversary $\mathcal{A}'$ that wins $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{Sh},\mathsf{Rec})}$ with non-negligible probability. We only sketch the proof. The reduction (that is, $\mathcal{A}'$) chooses a $\mathbf{S}_i \notin \mathcal{I}_0$ and a $\mathbf{R}_j \notin \mathcal{I}_j$, and simulates protocol $\Pi_{\mathsf{FutureMsg}}^k$ where the input of $\mathbf{S}_i$ for $\mathbf{R}_j$ is set (by $\mathcal{A}'$) to $m_{i,j}$. value $m_{i,j}$ to $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{Sh},\mathsf{Rec})}$, and then the t-shares to send to corrupted parties in $\mathcal{L}_{k'}$ are queried by $\mathcal{A}'$ from $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{Sh},\mathsf{Rec})}$. Then, the reduction continues to simulate the protocol and uses the shares received by $\mathbf{R}_j$ from corrupted parties $P_{k'}^\ell \in \mathcal{L}_{k'}$ as its final query to $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{Sh},\mathsf{Rec})}$. If $\mathcal{A}$ is such that the probability that the output of some honest $\mathbf{R}_j$ from an honest sender $\mathbf{S}_i$ is different in the real and ideal world, then

$$\mathsf{Adv}^{\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{Sh},\mathsf{Rec})}}(\mathcal{A}') = \frac{\delta}{\mathcal{O}(n^2)}. \tag{29}$$

□

5 Future Broadcast

5.1 Future Broadcast Functionality

This primitive allows a sender to broadcast any linear combination of some input values to a later layer (or one single party in a later layer), and guarantees that 1) the messages (or their wanted linear combination) remains secret until the decision to reveal them is taken, and 2) even when the sender is dishonest, all honest parties in the receiving layer agree on a single message. The decision to reveal a message (or not) can be taken by honest parties in a certain layer depending on public information. Again, this primitive provides no commitment guarantees when the sender is corrupt, as adversary is allowed to modify the message up until the moment of delivery.

Linear Future Broadcast Functionality $\mathcal{F}_{\mathsf{FutureBC}}^k$

Public Parameters. Sender $\mathbf{S} \in \mathcal{L}_0$. Auxiliary layer $\mathcal{L}_k$ deciding which messages are revealed. Layer $\mathcal{L}_{k'}$ onto which the messages are broadcast. The domain M of the messages from $\mathbf{S}$. The maximum number of messages ℓ to be broadcast by each sender.

Secret Inputs.

– Messages $(m_1, \ldots, m_\ell) \in M$ from $\mathbf{S}$ to be broadcast to $\mathcal{L}_{k'}$.

– A public value (L, r) agreed up on by all honest P_j^k, where

  - $L : M^\ell \to M$ is a linear operator.

- $r \in \mathcal{L}_{k'} \cup \{\mathcal{L}_{k'}\}$ is the intended recipient of $L(m_1, \ldots, m_\ell)$: either some specific party $P_s^{k'}$ in $\mathcal{L}_{k'}$ or all the parties in layer $\mathcal{L}_{k'}$.

Layer $\mathcal{L}_0$: If $\mathbf{S} \notin \mathcal{I}_0$ receive messages $(m_1, \ldots, m_\ell)$ from $\mathbf{S}$.

Layer $\mathcal{L}_k$: For each honest $P_i^k \in \mathcal{L}_k \setminus \mathcal{I}_k$, receive *the same* input (L, r).

Layer $\mathcal{L}_{k'-1}$: If $r = \mathcal{L}_{k'}$, forward $(L, L(m_1, \ldots, m_\ell))$ to the (ideal) adversary.

Layer $\mathcal{L}_{k'}$:

- If $r = P_s^{k'} \in \mathcal{I}_{k'}$, and $\mathbf{S} \notin \mathcal{I}_0$, forward $\big(L, L(m_1, \ldots, m_\ell)\big)$ to the (ideal) adversary.
- If $\mathbf{S} \in \mathcal{I}_0$ receive from the (ideal) adversary message $l_\mathcal{A}$ that $\mathbf{S}$ wants to broadcast.
- If $r = P_s^{k'} \notin \mathcal{I}_{k'}$, and $\mathbf{S} \notin \mathcal{I}_0$, send the value $L(m_1, \ldots, m_\ell)$ to $P_s^{k'}$.
- If $r = P_s^{k'} \notin \mathcal{I}_{k'}$, and $\mathbf{S} \in \mathcal{I}_0$, send the value $l_\mathcal{A}$ to $P_s^{k'}$.
- If $r = \mathcal{L}_{k'}$, and $\mathbf{S} \notin \mathcal{I}_0$, send the value $L(m_1, \ldots, m_\ell)$ to all parties in $\mathcal{L}_{k'}$.
- If $r = \mathcal{L}_{k'}$, and $\mathbf{S} \in \mathcal{I}_0$, send the value $l_\mathcal{A}$ to to all parties in $\mathcal{L}_{k'}$.

5.2 Future Broadcast Protocol

The first solution that comes to mind to realize $\mathcal{F}_{\mathsf{FutureBC}}$ is the following: to broadcast a message m onto layer $\mathcal{L}_{k'}$, simply provide, using future messaging, a robust sharing of s to layer $\mathcal{L}_k$, and ask parties in layer $\mathcal{L}_k$ to broadcast their shares using the provided broadcast channels. This construction is secure if there is only one recipient (notice that we do not provide any guarantees for dishonest recipients).

However, when the robust shares are broadcast, this construction is insecure, because *if the the dealer is honest* a rushing adversary can wait to see the shares broadcast by honest parties in $\mathcal{L}_k$ before broadcasting shares of corrupted parties. As mentioned in Section 2.4, the robustness guarantees provided by the secret sharing scheme are insufficient in this scenario, because the corrupted shares can depend on the honest shares. For instance, in the robust secret sharing scheme in Section 3.3, a rushing adversary would be able to make the reconstruction fail by first observing the keys broadcast by honest parties, and only then computing new valid MAC tags (with respect to these observed keys) for new arbitrary values.

To avoid this, we instead have the dealer set up n independent robust sharings of m and provide them to $\mathcal{L}_k$ using future messaging. Then, each of these states are reconstructed towards distinct parties in some auxiliary layer (no rushing attack applies if the recipient is honest), and then we leverage the honest majority in the auxiliary layer to agree on a single value: each party in the auxiliary layer simply broadcasts the value they have reconstructed. In this construction, the adversary learns the broadcasted value one layer before the intended target layer. Note, our functionality matches this protocol since the message is leaked to the adversary in the auxiliary layer.

We get around this shortcoming by ensuring that the adversary gains no advantage by having learned the broadcast message one layer in advance. This is arranged by having all the other protocols run in parallel with the future broadcast completely ignore the auxiliary layer. Consequently, the view of the adversary in any set of layers including the auxiliary layer and the output layer of a future broadcast is identical to that in the subset excluding the auxiliary layer. We formally describe the protocol below, and argue about its security, captured by Lemma 7, in Section 5.3.

Future Broadcast Protocol $\Pi^k_{\mathsf{FutureBC}}$

Public Parameters. Sender $\mathbf{S} \in \mathcal{L}_0$. Receiving layer $\mathcal{L}_k$. The domain M of $\mathbf{S}$'s messages.

Secret Inputs. Messages $(m_1, \ldots, m_\ell) \in M$ from sender $\mathbf{S}$.

Resources. A $(\mathcal{D}, t, \mathsf{negl}(\kappa))$-robust (t,n)-secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ with message space $\mathcal{M} = M$, randomness space $\mathcal{R}$ and share space $\mathcal{S}$; functionalities $\mathcal{F}^k_{\mathsf{FutureMsg}}$, $\mathcal{F}^{k'-k-1}_{\mathsf{FutureMsg}}$, and $\mathcal{F}^{k'-k}_{\mathsf{FutureMsg}}$.

Layer $\mathcal{L}_0$: $\mathbf{S} \in \mathcal{L}_0$ does

- Sample $(r_{1,j} \ldots, r_{\ell,j}) \leftarrow_{\mathcal{D}} \mathcal{R}$ for all $j \in [n]$.
- Compute $(m^1_{i,j}, \ldots, m^n_{i,j}) \leftarrow \mathsf{Sh}(m_i, r_{i,j})$ for each $i \in [\ell]$ and $j \in [n]$.
- For each $r \in [n]$, set m_r as the message [a] to P^k_r in $\mathcal{F}^k_{\mathsf{FutureMsg}}$, where m_r is the matrix of values $m^r_{i,j}$ for all $i \in [\ell]$ and $j \in [n]$.

Subroutine 1: Revealing $L(m_1, \ldots, m_\ell)$ to $P^{k'}_s$:

Layer $\mathcal{L}_k$: Each $P^k_r \in \mathcal{L}_k$ receives (from $\mathcal{F}^k_{\mathsf{FutureMsg}}$) values $\widehat{m^r_{i,j}}$ for all $i \in [\ell]$ and $j \in [n]$ and set input $m_{r,s}$ towards $P^{k'}_s$ in $\mathcal{F}^{(k'-k)}_{\mathsf{FutureMsg}}$ to the value $L\left(\widehat{m^r_{1,1}}, \ldots, \widehat{m^r_{\ell,1}}\right)$.

Layer $\mathcal{L}_{k'}$: Party $P^{k'}_s$ receives (from $\mathcal{F}^{k'-k}_{\mathsf{FutureMsg}}$) values $\widehat{l^r}$ for all $r \in [n]$ and computes their output $m_s \leftarrow \mathsf{Rec}\left(\widehat{l^1}, \ldots, \widehat{l^n}\right)$.

Subroutine 2: Revealing $L(m_1, \ldots, m_\ell)$ to all parties in $\mathcal{L}_{k'}$:

Layer $\mathcal{L}_k$: Each $P^k_r \in \mathcal{L}_k$ receives (from $\mathcal{F}^k_{\mathsf{FutureMsg}}$) values $\widehat{m^r_{i,j}}$ for all $i \in [\ell]$ and $j \in [n]$ and sets input $m_{r,j}$ towards $P^{k'-1}_j$ in $\mathcal{F}^{(k'-k-1)}_{\mathsf{FutureMsg}}$ to $L\left(\widehat{m^r_{1,j}}, \ldots, \widehat{m^r_{\ell,j}}\right)$.

Layer $\mathcal{L}_{k'-1}$: Each $P_j^{k'-1}$ receives (from $\mathcal{F}_{\mathsf{FutureMsg}}^{k'-k-1}$) values $\widehat{l_j^r}$ for all $r \in [n]$ and broadcasts (using the provided ideal broadcast channels) value $l_j \leftarrow \mathsf{Rec}\left(\widehat{l_j^1}, \ldots, \widehat{l_j^n}\right)$.

Layer $\mathcal{L}_{k'}$: Each party in $\mathcal{L}_{k'}$ outputs the value which was broadcast the most times by $\mathcal{L}_{k'-1}$.

[a]Since we are using the parallel $\mathcal{F}_{\mathsf{FutureMsg}}$ functionality with only one sender we drop the indices for clarity.

5.3 Future Broadcast Security

Lemma 7. *If $t < n/2$ and $(\mathsf{Sh}, \mathsf{Rec})$ is a $(\mathcal{D}, t, \mathsf{negl}(\kappa))$-robust (t, n)-secret-sharing scheme, then for all $k \in [k']$ the (n, t, k')-layered protocol $\Pi_{\mathsf{FutureBC}}^{k,k'}$ realizes $\mathcal{F}_{\mathsf{FutureBC}}^{k,k'}$ with $(\mathsf{negl}(\kappa), t)$-statistical security in the $\mathcal{F}_{\mathsf{FutureMsg}}$-hybrid model.*

Proof. For any adversary $\mathcal{A}$ we describe a simulator σ such that the joint distribution of the outputs of the adversary and honest parties in the real (hybrid) world (the adversary $\mathcal{A}$ interacting with protocol $\Pi_{\mathsf{FutureMsg}}^k$ and functionalities $\mathcal{F}_{\mathsf{FutureMsg}}^k, \mathcal{F}_{\mathsf{FutureMsg}}^{k'-k}, \mathcal{F}_{\mathsf{FutureMsg}}^{k'-k-1}$) is statistically close to that in the ideal world (the simulator σ interacting with functionality $\mathcal{F}_{\mathsf{FutureBC}}^{k,k'}$). We will show that if this statistical distance is non-negligible, then we can produce an adversary $\mathcal{A}'(\mathcal{A})$ that wins game $\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{Sh},\mathsf{Rec})}$ with non-negligible probability, therefore reducing the security of $\Pi_{\mathsf{FutureBC}}^k$ in the $\left(\mathcal{F}_{\mathsf{FutureMsg}}^k, \mathcal{F}_{\mathsf{FutureMsg}}^{k'-k}, \mathcal{F}_{\mathsf{FutureMsg}}^{k'-k-1}\right)$-hybrid model to the $(\mathcal{D}, \ell, \delta)$-robustness of $(\mathsf{Sh}, \mathsf{Rec})$. First, given any adversary $\mathcal{A}$ interacting with protocol $\Pi_{\mathsf{FutureBC}}^k$ we describe the simulator σ.

Simulator σ for Protocol $\Pi_{\mathsf{FutureBC}}^{k,k'}$

Layer $\mathcal{L}_0$: If $\mathbf{S} \notin \mathcal{I}_0$

- Sample $(r_{1,j} \ldots, r_{\ell,j}) \leftarrow_{\mathcal{D}} \mathcal{R}$ for all $j \in [n]$.
- Compute $\left(m_{i,j}^1, \ldots, m_{i,j}^n\right) \leftarrow \mathsf{Sh}(0, r_{i,j})$ for all $i \in [\ell]$ and $j \in [n]$.

Layer $\mathcal{L}_k$:

- If $\mathbf{S} \notin \mathcal{I}_0$, for all $P_r^k \in \mathcal{I}_k$ send $\mathcal{A}$ the matrix of values $m_{i,j}^r$ for all $i \in [\ell]$ and $j \in [n]$ as output from $\mathcal{F}_{\mathsf{FutureMsg}}^k$.

- If $\mathbf{S} \in \mathcal{I}_0$ receive from $\mathcal{A}$ values $\left(\widehat{m_{i,j}^1}, \ldots, \widehat{m_{i,j}^n}\right)$ for all $i \in [\ell]$ and $j \in [n]$ as inputs to $\mathcal{F}_{\mathsf{FutureMsg}}^k$.

Layer $\mathcal{L}_{k'-1}$: If $\mathbf{S} \notin \mathcal{I}_0$

- Receive $(L, L(m_1, \ldots, m_\ell))$ (possibly $\perp$) from $\mathcal{F}_{\mathsf{FutureBC}}^{k,k'}$.

- For all $P_r^k \in \mathcal{I}_k$ compute $l_j^r = \left(m_{1,j}^r, \dots, m_{\ell,j}^r\right)$ for all $j \in [n]$.
- For all $j \in [n]$ patch the simulated honest states to
$$\left(\left(l_j^r\right)_{r \in \mathcal{I}_k}, \left(\widehat{l_j^r}\right)_{r \notin \mathcal{I}_k}\right) \tag{30}$$
to obtain a valid sharing of $l_j = L'(m_1, \dots, m_\ell)$, as explained in Lemma 5.
- For each $P_r^k \notin \mathcal{I}_k$ and all $j \in [n]$ send $\widehat{l_j^r}$ to $\mathcal{A}$ as output from $\mathcal{F}_{\mathsf{FutureMsg}}^{k'-k-1}$.
- For all $P_r^k \in \mathcal{I}_k$ receive from $\mathcal{A}$ values $\widehat{l_j^r}$ as input to $\mathcal{F}_{\mathsf{FutureMsg}}^{k'-k-1}$ for $P_j^{k'-1}$.
- For all honest $P_j^{k'-1}$ compute
$$l_j = \mathsf{Rec}\left(\left(\widehat{l_j^r}\right)_{r \in \mathcal{I}_k}, \left(\widehat{l_j^r}\right)_{r \notin \mathcal{I}_k}\right), \tag{31}$$
and broadcast l_j on behalf of $P_j^{k'-1}$.

Layer $\mathcal{L}_{k'}$: If $\mathbf{S} \notin \mathcal{I}_0$

- Receive $(L', L'(m_1, \dots, m_\ell))$ from $\mathcal{F}_{\mathsf{FutureBC}}^{k,k'}$ (possibly $\perp$) from $\mathcal{F}_{\mathsf{FutureBC}}^{k,k'}$.
- For all $P_r^k \in \mathcal{I}_k$ compute ${l'}_1^r = \left(m_{1,j}^r, \dots, m_{\ell,j}^r\right)$ for all $j \in [n]$.
- Patch the simulated honest state
$$\left(\left({l'}_1^r\right)_{r \in \mathcal{I}_k}, \left(\widehat{{l'}_1^r}\right)_{r \notin \mathcal{I}_k}\right). \tag{32}$$
to obtain a valid sharing of $l'_1 = L'(m_1, \dots, m_\ell)$, as explained in Lemma 5.
- For each $P_r^k \notin \mathcal{I}_k$ send $\widehat{l_j^r}$ to $\mathcal{A}$ as output from $\mathcal{F}_{\mathsf{FutureMsg}}^{k'-k}$.
- For all $P_r^k \in \mathcal{I}_k$ receive from $\mathcal{A}$ values $\widehat{l_j^r}$ as inputs to $\mathcal{F}_{\mathsf{FutureMsg}}^{k'-k}$ for $P_j^{k'}$.
- For each $P_j^{k'-1} \in \mathcal{I}_{k'-1}$ receive broadcast values l_j from $\mathcal{A}$.
- If $\mathbf{S} \in \mathcal{I}_0$ and $L \neq \perp$ compute $l_{\mathcal{A}}$ as any value appearing at least $t+1$ times among values $(l_1, \dots, l_n)$, and input $l_{\mathcal{A}}$ to $\mathcal{F}_{\mathsf{FutureBC}}^{k}$.
- If $\mathbf{S} \in \mathcal{I}_0$ and $L' \neq \perp$ compute
$$l'_{\mathcal{A}} = \mathsf{Rec}\left(\left(\widehat{{l'}_1^r}\right)_{r \in \mathcal{I}_k}, \left(\widehat{{l'}_1^r}\right)_{r \notin \mathcal{I}_k}\right) \tag{33}$$
and input $l'_{\mathcal{A}}$ to $\mathcal{F}_{\mathsf{FutureBC}}^{k}$.
- Set output to the output of $\mathcal{A}$.

Let's start by arguing about the view of the adversary. By inspection of the protocol, the only difference between the view of $\mathcal{A}$ in the real and ideal world is that when the sender $\mathbf{S}$ is honest, in the ideal world values $m_{i,j}^r$ for $P_r^k \in \mathcal{I}_k$ that $\mathcal{A}$ receives in layer $\mathcal{L}_k$ are sampled as $\left(m_{i,j}^1, \ldots, m_{i,j}^n\right) \leftarrow \mathsf{Sh}(0, r_{i,j})$ for all $i \in [\ell]$ and $j \in [n]$, while in the real world they are sampled as $\left(m_{i,j}^1, \ldots, m_{i,j}^n\right) \leftarrow \mathsf{Sh}(m_i, r_{i,j})$ for all $i \in [\ell]$ and $j \in [n]$, where m_i for $i \in [\ell]$ are the real inputs of $\mathbf{S}$. However, since $|\mathcal{I}_k| \leq t$, by t-privacy of $(\mathsf{Sh}, \mathsf{Rec})$ the distribution of up to t shares is identical regardless of the secret. To continue, we distinguish two cases: if all honest parties execute Subroutine 1 (private reveal of $L(m_1, \ldots, m_\ell)$ to $P_s^{k'}$), then both in the real and ideal world the adversary does not learn any further values if $P_s^{k'}$ is honest. If $P_s^{k'}$ is corrupt, the adversary observes shares $\widehat{l'^r_1}$ for all honest $P_r^k \notin \mathcal{I}_k$ in the ideal world, while it observes values $L\left(\widehat{m_{1,1}^r}, \ldots, \widehat{m_{n,1}^r}\right)$ for all honest $P_r^k \notin \mathcal{I}_k$ in the real world if the sender is honest (the case where the sender is corrupt is trivial as all values $\mathcal{A}$ receives here are sampled from $\mathcal{A}$). Thanks to the correct patching of $(\mathsf{Sh}, \mathsf{Rec})$ and its $\mathcal{D}$-linearity, in both scenarios the adversary receives $t+1$ shares from honestly sampled sharings of $L(m_1, \ldots, m_\ell)$, so that the view of the adversary is identically distributed in the real and ideal world. When honest parties execute Subroutine 2 the argument is analogous, even though when $\mathbf{S}$ is honest then $\mathcal{A}$ learns $t+1$ shares from each of $t+1$ independent and honestly generated sharings of $L(m_1, \ldots, m_\ell)$. Now let's argue about the output of honest parties in $\mathcal{L}_{k'} \setminus \mathcal{I}_{k'}$. If all honest parties in layers $\mathcal{L}_k$ to $\mathcal{L}_{k'}$ execute Subroutine 1 with the same linear function L and towards the same party $P_s^{k'}$, then $P_s^{k'}$ is the only honest party with output. If the sender $\mathbf{S}$ is corrupted, then the output of $P_s^{k'}$ is calculated in the same way in the real and in the ideal world. Now, if $\mathbf{S}$ is honest, in the ideal world the output of $P_s^{k'}$ is simply $L(m_1, \ldots, m_\ell)$ of the inputs of the sender. In the real world the output of $P_s^{k'}$ is $m_s = \mathsf{Rec}\left(\widehat{l^1}, \ldots, \widehat{l^n}\right)$, where $\widehat{l^r}$ is the message received from $P_s^{k'}$ in $\mathcal{F}_{\mathsf{FutureMsg}}^{k'-k}$. Because at least $n-t$ parties in $\mathcal{L}_k \setminus \mathcal{I}_k$ are honest, then at least $n-t$ of values $\widehat{l^r}$ are equal to $L\left(\widehat{m_{1,1}^r}, \ldots, \widehat{m_{\ell,1}^r}\right)$. Therefore, If $\mathbf{S}$ is honest, $\widehat{m_{i,1}^r}$ for $r \in [n]$ is a correct robust sharing of m_i. If with non-negligible probability the the output of P_s^k in the real world is not $L(m_1, \ldots, m_\ell)$ (let's say with probability δ) we can easily produce an adversary $\mathcal{A}'$ such that

$$\mathsf{Adv}^{\mathsf{Rob}_{\mathcal{D},q}^{(\mathsf{Sh},\mathsf{Rec})}}(\mathcal{A}') = \frac{\delta}{\mathcal{O}(n)}. \tag{34}$$

Analyzing the outputs of honest parties in Subroutine 2 is analogous: if the (unique) output of honest parties is different in the real and ideal world with non-negligible probability, by leveraging the honest majority in $\mathcal{L}_{k'-1}$, this means that with non negligible probability the adversary breaks the robustness of the underlying secret sharing scheme for at least one honest party in $\mathcal{L}_{k'-1}$. We omit the details.

□

6 Information Theoretic Signature

6.1 Information Theoretic Signature Functionality

This functionality allows a sender to entrust an intermediary with a message that a later layer can then reliably reveal to a receiver. The functionality ensures that an honest intermediary will always be able to convince the honest receiver to accept the message that the intermediary received from the sender. If the sender is honest, the functionality ensures that a potentially corrupt intermediary cannot convince an honest receiver to accept a distinct message than the one it received from the sender. When both the sender and intermediary are corrupt, the functionality provides no guarantees. Notice that in the layered setting, the message is not *actually* revealed by the intermediary itself, but rather by a later layer holding an appropriate state provided by the intermediary.

The functionality described below describes a generalization useful in our later constructions in which the message receiver(s) can be selected from a set of possible layers $\{\mathcal{L}_{r_1}, \ldots, \mathcal{L}_{r_v}\}$ by one among a set of auxiliary layers $\{\mathcal{L}_{k_1}, \ldots, \mathcal{L}_{k_v}\}$. Note, the sender is forced to commit to the message by the first among the auxiliary layers.

Information Theoretic Signature Functionality $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$

Public Parameters. Sender $\mathbf{S} \in \mathcal{L}_0$. Intermediary $\mathbf{M} \in \mathcal{L}_k$ for $k \geq 1$. Auxiliary layers $\mathcal{L}_{k_1}, \ldots, \mathcal{L}_{k_w}$ for $k_w > \ldots > k_1 \geq k + 8$. Candidate receiver layers $\mathcal{L}_{r_1}, \ldots, \mathcal{L}_{r_v}$, $r_1 > k_1$. The $\mathbb{F}$-vector space M, domain of $\mathbf{S}$'s messages.

Inputs. From $\mathbf{S}$, secret messages $(m_1, \ldots, m_\ell) \in M^\ell$. The same public input (L, r) from all honest parties from a unique auxiliary layer $\mathcal{L}_{k_i}$, $i \in [w]$.

- $L : M^\ell \to M$ is the linear operator.
- $r \in \mathcal{L}_{r_i} \cup \{\mathcal{L}_{r_i}\}$ for some $i \in [v]$ is the intended recipient of $L(m_1, \ldots, m_\ell)$: either a specific party in $\mathcal{L}_{r_i}$ or all parties in layer $\mathcal{L}_{r_i}$.

Layer $\mathcal{L}_0$:

- If $\mathbf{S} \notin \mathcal{I}_0$, receive messages $(m_1, \ldots, m_\ell)$ from $\mathbf{S}$.

Layer $\mathcal{L}_k$:

- If $\mathbf{M} \in \mathcal{I}_k$ is corrupt, reveal $(m_1, \ldots, m_\ell)$ to the (ideal) adversary.

Layer $\mathcal{L}_{k_1}$: If $\mathbf{M} \notin \mathcal{I}_k$ is honest and $\mathbf{S} \in \mathcal{I}_0$ is corrupt, receive messages $(m_1, \ldots, m_\ell)$ from the (ideal) adversary.

Layer $\mathcal{L}_{k_i}$ for $i \in [w]$:

- Receive (the same) message (r, L) or $\perp$ from each honest party in $\mathcal{L}_{k_i}$. Ignore messages from $\mathcal{L}_{k_{i'}}, i' > i$ if (r, L) is received in $\mathcal{L}_{k_i}$.

Layer $\mathcal{L}_{r_i - 3}$ for $i \in [v]$:

- If $r = \mathcal{L}_{r_i}$ (i.e., specifically, the set of receivers is the entire $\mathcal{L}_{r_i}$ instead of an individual receiver $P_j^{r_i} \in \mathcal{L}_{r_i}$ for some $j \in [n]$), and both $\mathbf{M} \notin \mathcal{I}_k$ and $\mathbf{S} \notin \mathcal{I}_0$ are honest, deliver $L(m_1, \ldots, m_\ell)$ to the ideal adversary.

Layer $\mathcal{L}_{r_i}$ for $i \in [v]$:

1. If $r = \mathcal{L}_{r_i}$ or $r = P_j^{r_i} \in \mathcal{L}_{r_i}$:
   - If $\mathbf{M}$ is honest deliver $L(m_1, \ldots, m_\ell)$ to r.
   - If $\mathbf{M}$ is corrupt then:
     - If $\mathbf{S}$ is corrupt receive m' from the (ideal) adversary and forward it to r.
     - If $\mathbf{S}$ is honest receive boolean $\mathsf{reveal} \in \{0, 1\}$ from the (ideal) adversary. If $\mathsf{reveal} = 1$ then deliver $L(m_1, \ldots, m_\ell)$ to r.

6.2 Information Theoretic Signature Protocol

For improved legibility, we describe a protocol realizing the above functionality for $\ell = 1$, $w = 1$ and $v = 1$, meaning the auxiliary layer is fixed to $\mathcal{L}_{k'}$ and receiver layer is $\mathcal{L}_{k''}$. We only consider the more involved construction in which the message is revealed to *all* the parties in $\mathcal{L}_{k''}$. We will describe later how the protocol can be modified to realize the general functionality for arbitrary, finite ℓ, w and v. Section 2.5 contains a more detailed description of the protocol.

Information Theoretic Signature Protocol $\Pi_{\mathsf{ITSig}}^{k'}$

Public Parameters. Sender $\mathbf{S} \in \mathcal{L}_0$, intermediary $\mathbf{M} \in \mathcal{L}_k$, a committing layer k' and receiver layer $\mathbf{R} = \mathcal{L}_{k''}$ where $1 \leq k < k' < k''$. The message domain M which is a finite field. A security parameter κ.

Secret Inputs. $\mathbf{S}$ has a message $m \in M$ to be sent to $\mathbf{R}$ via $\mathbf{M}$.

Resources. Functionality $\mathcal{F}_{\mathsf{FutureMsg}}$; functionality $\mathcal{F}_{\mathsf{FutureBC}}$; a message authentication code in which key $(a, b) \leftarrow_\$ M^2$ and $\mathsf{Aut}(m, (a, b)) = a \cdot m + b$ for any message in M.

Layer $\mathcal{L}_0$: The sender $\mathbf{S}$ does:

1. Sample keys $k_{i,j} \leftarrow_\$ \mathbb{F}^2$ for each $i \in [n]$, $j \in [\kappa]$, and compute $t_{i,j} = \mathsf{Aut}(k_{i,j}, m)$ for all $i \in [n]$ and $j \in [\kappa]$.

2. Send $(m, \{t_{i,j}\}_{i\in[n],j\in[\kappa]})$ to $\mathbf{M} \in \mathcal{L}_k$ using $\mathcal{F}_{\mathsf{FutureMsg}}$.
3. Send $\{k_{i,j}\}_{i\in[n],j\in[\kappa]}$ to each $P_i \in \mathcal{L}_{k+1}$ using $\mathcal{F}_{\mathsf{FutureMsg}}$ for all $i \in [n]$.
4. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with $k_{i,j}$ as input and $\mathcal{L}_{k+2}$ for all $i \in [n]$ and $j \in [\kappa]$.
5. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with m as input and $\mathcal{L}_{k+6}$ as auxiliary layer.

Layer $\mathcal{L}_k$: The intermediary $\mathbf{M}$ does:

1. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with input $(m, t_{i,j})$ and $\mathcal{L}_{k+4}$ as auxiliary layer for all $i \in [n]$ and $j \in [\kappa]$.
2. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with input m and $\mathcal{L}_{k'}$ as auxiliary layer.
3. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with input $t_{i,j}$ and $\mathcal{L}_{k'}$ as auxiliary layer for all $i \in [n]$ and $j \in [\kappa]$.

Layer $\mathcal{L}_{k+1}$: Each $P_i^{k+1} \in \mathcal{L}_{k+1}$ does:

1. Choose a random subset $\mathrm{IND}_i \subset [\kappa]$ of size $\kappa/2$, and broadcasts IND_i.
2. Broadcast $k_{i,j}$ for all $j \in \mathrm{IND}_i$
3. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with $k_{i,j}$ as input and $\mathcal{L}_{k+4}$ as auxiliary layer for all $j \notin \mathrm{IND}_i$.

Layer $\mathcal{L}_{k+2}$:

1. For each $i \in [n], j \in \mathrm{IND}_i$, to $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{S}$ as sender, $k_{i,j}$ as message, and $\mathcal{L}_{k+2}$ as auxiliary layer, all parties send $(L \leftarrow (1), r \leftarrow \mathcal{L}_{k+4})$ as input.

Layer $\mathcal{L}_{k+4}$:

1. For each $i \in [n], j \in \mathrm{IND}_i$, each party stores $k_{i,j}$ sent by $\mathbf{S}$ using $\mathcal{F}_{\mathsf{FutureBC}}$ as $\bar{k}_{i,j}$, and that sent by $P_i \in \mathcal{L}_{k+2}$ as $\tilde{k}_{i,j}$.
2. Each party computes and broadcasts
$$\mathtt{votes} = \{i \in [n] : \exists j \in \mathrm{IND}_i, \bar{k}_{i,j} \neq \tilde{k}_{i,j}\}. \tag{35}$$
3. For each $i \in [n], j \in \mathrm{IND}_i$, invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, $(m, t_{i,j})$ as message, and $\mathcal{L}_{k+4}$ as auxiliary layer, all parties send $(L \leftarrow (a_{i,j}, b_{i,j}), r \leftarrow \mathcal{L}_{k+4})$, where $(a_{i,j}, b_{i,j}) = \bar{k}_{i,j}$, as input.
4. For each $i \notin \mathtt{votes}, j \notin \mathrm{IND}_i$, invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with $P_i \in \mathcal{L}_{k+1}$ as sender, $k_{i,j}$ as message, and $\mathcal{L}_{k+4}$ as auxiliary layer, all parties send $(L \leftarrow (1), r \leftarrow \mathcal{L}_{k''})$ as input.

Layer $\mathcal{L}_{k+6}$:

1. If there exists $i \in [n]$ and $j \in \text{IND}_i$ such that the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender and $(m, t_{i,j})$ as message, is non-zero, then set $\texttt{success} = 0$, otherwise set it to 1. Broadcast $\texttt{success}$.

2. If $\texttt{success} = 0$, call $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{S}$ as sender, m as message, and $\mathcal{L}_{k+6}$ as auxiliary layer, parties send $(L \leftarrow (1), r \leftarrow \mathcal{L}_{k'})$ as input.

Layer $\mathcal{L}_{k'}$:

1. If $\texttt{success} = 0$, each party receives m' as the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{S}$ as sender, m as message, and $\mathcal{L}_{k+6}$ as auxiliary layer. Each party broadcasts m'.

2. If $\texttt{success} = 1$:

   (a) Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, m as message, and $\mathcal{L}_{k'}$ as auxiliary layer, parties send $(L \leftarrow (1), r \leftarrow \mathcal{L}_{k''-2})$ as input.

   (b) For each $i \notin \texttt{votes}, j \notin \text{IND}_i$, to $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, $t_{i,j}$ as message, and $\mathcal{L}_{k'}$ as auxiliary layer, parties send $(L \leftarrow (1), r \leftarrow \mathcal{L}_{k''-2})$ as input.

Layer $\mathcal{L}_{k''-2}$:

1. Each party recovers $\tilde{m}$ as the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, m as message, and $\mathcal{L}_{k'}$ as auxiliary layer.

2. For each $i \notin \texttt{votes}$ and $j \notin \text{IND}_j$, each party recovers $\tilde{t}_{i,j}$ as the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, $t_{i,j}$ as message, and $\mathcal{L}_{k'}$ as auxiliary layer.

3. Each party broadcasts $\tilde{m}$ and $\{\tilde{t}_{i,j}\}_{i \in [n], j \in \text{IND}_i}$.

Layer $\mathcal{L}_{k''}$:

1. If $\texttt{success} = 0$, the parties output m' broadcasted by parties in $\mathcal{L}_{k'}$.

2. If $\texttt{success} = 1$:

   (a) For each $i \notin \texttt{votes}, j \notin \text{IND}_i$, each party recovers $\tilde{k}_{i,j}$ as the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $P_i \in \mathcal{L}_{k+1}$ as sender, $k_{i,j}$ as message, and $\mathcal{L}_{k+4}$ as auxiliary layer

   (b) For each $i \notin [\texttt{votes}]$, and $j \notin \text{IND}_i$, using $\tilde{m}$ and $\tilde{t}_{i,j}$ broadcast by the $\mathcal{L}_{k''-2}$, and $\tilde{k}_{i,j}$, each party checks if $\mathsf{Vfy}(\tilde{t}_{i,j}, \tilde{m}, \tilde{k}_{i,j}) = 1$; if so, $\texttt{votes} \leftarrow \texttt{votes} \cup \{i\}$.

   (c) If $|\texttt{votes}| \geq t + 1$, each party outputs $\tilde{m}$; else outputs $\perp$.

6.3 Information Theoretic Signatures Security

Lemma 8. *If $t < n/2$ and $(\mathsf{Aut}, \mathsf{Vfy})$ is a $(\mathcal{D}, t, \mathsf{negl}(\kappa))$-secure MAC, then the (n, t, k'')-layered protocol Π_{ITSig} realizes functionality $\mathcal{F}_{\mathsf{ITSig}}$ with $\ell = 1$, $w = 1$ and $v = 1$ with $(\mathsf{negl}(\kappa), t)$-statistical security in the $(\mathcal{F}_{\mathsf{FutureMsg}}, \mathcal{F}_{\mathsf{FutureBC}})$-hybrid model.*

Proof. We describe a simulator σ such that, for any adversary $\mathcal{A}$, the joint distribution of the outputs of the adversary and honest parties in the real world where the adversary $\mathcal{A}$ interacts with protocol Π_{ITSig} in the $(\mathcal{F}_{\mathsf{FutureMsg}}, \mathcal{F}_{\mathsf{FutureBC}})$-hybrid world is statistically close to that in the ideal world interaction between simulator $\sigma(\mathcal{A})$ and functionality $\mathcal{F}_{\mathsf{ITSig}}$.

Simulator $\sigma(\mathcal{A})$ for protocol Π_{ITSig}

For clarity in presentation, the behavior of the simulator is described separately for four different scenarios based on the corruption status of $\mathbf{S}$ and $\mathbf{M}$. We stress that, despite this, the behavior of the simulator in a layer does not depend on the set of corrupt parties in the subsequent layers.

Case: $\mathbf{S} \in \mathcal{I}_0$ and $\mathbf{M} \in \mathcal{I}_k$

$\mathbf{S}$ is the only party with input. Hence, the simulator $\sigma(\mathcal{A})$ emulates all the honest parties and the ideal functionalities $\mathcal{F}_{\mathsf{FutureMsg}}$ and $\mathcal{F}_{\mathsf{FutureBC}}$, and interacts with the adversary $\mathcal{A}$. In $\mathcal{L}_{k''}$, let m' be the output computed by of one of the emulated honest parties. $\sigma(\mathcal{A})$ sends m' to $\mathcal{F}_{\mathsf{ITSig}}$ in $\mathcal{L}_{k''}$. Finally, $\sigma(\mathcal{A})$ outputs whatever $\mathcal{A}$ outputs in the interaction.

Case: $\mathbf{S} \in \mathcal{I}_0$ and $\mathbf{M} \notin \mathcal{I}_k$

$\mathbf{S}$ is the only party with input. Hence, once again, the simulator $\sigma(\mathcal{A})$ emulates all the honest parties and the ideal functionalities $\mathcal{F}_{\mathsf{FutureMsg}}$ and $\mathcal{F}_{\mathsf{FutureBC}}$, and interacts with the adversary $\mathcal{A}$. In $\mathcal{L}_{k'}$, suppose one of the emulated honest parties, say $P_i \in \mathcal{L}_{k'}$, has set $\texttt{success} = 0$. Then, $\sigma(\mathcal{A})$ sends m' computed by P_i (see $\mathcal{L}_{k'}$ step 1 of the protocol) to $\mathcal{F}_{\mathsf{ITSig}}$ in $\mathcal{L}_{k'}$. Otherwise, $\sigma(\mathcal{A})$ sends m to $\mathcal{F}_{\mathsf{ITSig}}$ in $\mathcal{L}_{k'}$, where m is the message that $\mathcal{A}$ sends to emulated $\mathbf{M}$ on behalf of the corrupt sender (see $\mathcal{L}_0$ step 2). Finally, $\sigma(\mathcal{A})$ outputs whatever $\mathcal{A}$ outputs in the interaction.

Case: $\mathbf{S} \notin \mathcal{I}_0$ and $\mathbf{M} \in \mathcal{I}_k$

In $\mathcal{L}_k$, simulator $\sigma(\mathcal{A})$ learns that $\mathbf{M} \in \mathcal{I}_k$. $\sigma(\mathcal{A})$ receives m from $\mathcal{F}_{\mathsf{ITSig}}$. At this point, $\sigma(\mathcal{A})$ emulates $\mathbf{S}$ with input m, all the honest parties in $\mathcal{L}_k, \ldots, \mathcal{L}_{k''}$ and the ideal functionalities $\mathcal{F}_{\mathsf{FutureMsg}}$ and $\mathcal{F}_{\mathsf{FutureBC}}$, and interacts with $\mathcal{A}$. Here, we crucially used the fact that, delaying the emulation of $\mathbf{S}$ till $\mathcal{L}_k$ does not affect the simulation since no party in $\mathcal{L}_1, \ldots, \mathcal{L}_k - 1$ receive any message in $(\mathcal{F}_{\mathsf{FutureMsg}}, \mathcal{F}_{\mathsf{FutureBC}})$-hybrid model.
In $\mathcal{L}_{k''}$, let m' be the output computed by of one of the emulated honest parties. $\sigma(\mathcal{A})$ sends m' to $\mathcal{F}_{\mathsf{ITSig}}$. Finally, $\sigma(\mathcal{A})$ outputs whatever $\mathcal{A}$ outputs in the interaction.

Case: $\mathbf{S} \notin \mathcal{I}_0$ and $\mathbf{M} \notin \mathcal{I}_k$

In $\mathcal{L}_k$, simulator $\sigma(\mathcal{A})$ learns that $\mathbf{M} \notin \mathcal{I}_k$. At this point, $\sigma(\mathcal{A})$ emulates $\mathbf{S}$ with default input 0, all the honest parties in $\mathcal{L}_k, \ldots, \mathcal{L}_{k''}$ and the ideal functionalities $\mathcal{F}_{\mathsf{FutureMsg}}$ and $\mathcal{F}_{\mathsf{FutureBC}}$, and interacts with $\mathcal{A}$. Once again, we crucially used the fact that, delaying the emulation of $\mathbf{S}$ till $\mathcal{L}_k$ does not affect the simulation since no party in $\mathcal{L}_1, \ldots, \mathcal{L}_{k-1}$ receive any message (from the sender) in $\mathcal{F}_{\mathsf{FutureMsg}}$ and $\mathcal{F}_{\mathsf{FutureBC}}$ hybrid model.
In $\mathcal{L}_{k''-3}$, receive m from $\mathcal{F}_{\mathsf{ITSig}}$. From the states of an emulated honest party, $\sigma(\mathcal{A})$ extracts `votes` and IND_i for each $i \notin$ `votes`. Then, behaves as follows:

1. $\sigma(\mathcal{A})$ reports m as the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, m as message, and $\mathcal{L}_{k'}$ as auxiliary layer (see steps 1.(a) in $\mathcal{L}_k$, 2. (a) in $\mathcal{L}_{k'}$, and 1 in $\mathcal{L}_{k''-2}$).

2. For each $i \notin$ `votes` and $j \notin \mathrm{IND}_i$, let $K_{i,j}$ be the value sampled by $\mathbf{S}$ (see step 1 in $\mathcal{L}_0$), then $\sigma(\mathcal{A})$ computes $M_{i,j} = \mathsf{Aut}(m, K_{i,j})$, and reports $M_{i,j}$ as the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, $M_{i,j}$ as message, and $\mathcal{L}_{k'}$ as auxiliary layer (see steps 1.(b) in $\mathcal{L}_k$, 2. (b) in $\mathcal{L}_{k'}$, and 2 in $\mathcal{L}_{k''-2}$).

Finally, $\sigma(\mathcal{A})$ outputs whatever $\mathcal{A}$ outputs in the interaction.

We will separately consider the four different scenarios considered in the simulation.

Case: $\mathbf{S} \in \mathcal{I}_0$ and $\mathbf{M} \in \mathcal{I}_k$. In this scenario, the real and ideal executions are identical but with exactly one distinction: the output of the honest parties in the latter is m provided by $\mathcal{F}_{\mathsf{ITSig}}$ where m is the output of one of the honest receivers emulated by $\sigma(\mathcal{A})$. Clearly, the joint distribution of the output of the simulator and the outputs of the honest receivers emulated by $\sigma(\mathcal{A})$ in the ideal execution is identical to that of the output of $\mathcal{A}$ and the output of the honest parties in the real execution. But, the output of all honest parties are equal since their views are identical; this follows from the fact that every message received by the receivers in $\mathcal{L}_{k'}$ is a broadcast message.

Case: $\mathbf{S} \in \mathcal{I}_0$ and $\mathbf{M} \notin \mathcal{I}_k$. In this scenario also, the real and ideal executions are identical but with exactly one distinction: the output of the honest parties in the latter is m provided by $\mathcal{F}_{\mathsf{ITSig}}$, where m is chosen as follows by $\sigma(\mathcal{A})$:

- In $\mathcal{L}_{k'}$, suppose one of the emulated honest parties, say $P_i \in \mathcal{L}_{k'}$ has set `success` $= 0$. Then, $m = m'$ computed by P_i (see $\mathcal{L}_{k'}$ step 1 of the protocol).

- Otherwise, m is the message that $\mathcal{A}$ sends to emulated $\mathbf{M}$ on behalf of the corrupt sender (see $\mathcal{L}_0$ step 2).

The joint distribution of the output of the simulator and the outputs of the honest receivers emulated by $\sigma(\mathcal{A})$ in the ideal execution is identical to that of the output of $\mathcal{A}$ and the output of the honest parties in the real execution. Hence, it suffices to show that, with overwhelming probability, the outputs of the all the honest receivers emulated by $\sigma(\mathcal{A})$ coincide with m'.

First, consider the event where an honest party in $\mathcal{L}_{k'}$ has set `success` $= 0$. In this event, all honest parties in the layer would set `success` $= 0$ since this value is received by the parties over a broadcast. It can be verified by inspection that the output of all honest parties is m as computed by all honest parties in step 1 of $\mathcal{L}_{k'}$. Thus, in this event, the outputs of the all the honest receivers emulated by $\sigma(\mathcal{A})$ coincide with m'.

Next, suppose $\texttt{success} = 1$. In this event, we claim that, for each honest $P_i \in \mathcal{L}_{k+1}$, there exists $j \notin \text{IND}_i$ such that $K_{i,j}$ received by P_i from $\mathbf{S}$ and m and $M_{i,j}$ received by $\mathbf{M}$ from $\mathbf{S}$ satisfy $M_{i,j} = \mathsf{Aut}(m, K_{i,j})$. Assuming this holds, we claim that $i \in \texttt{votes}$ whenever $P_i \in \mathcal{L}_k$ is honest. This is argued as follows: suppose i is not added to $\texttt{votes}$ in step 2 of $\mathcal{L}_{k+4}$. Then, in step 2.(b), i will be added to $\texttt{votes}$ by our assumption. Hence, $|\texttt{votes}| \geq t + 1$. Therefore, all honest parties output the message m that $\mathbf{M}$ received from $\mathbf{S}$.

We conclude the proof by proving our assumption. Fix i such that $P_i \in \mathcal{L}_{k+1}$ is honest. When $m, M_{i,j}$ and $K_{i,j}$ for $j \in [\kappa]$ are as described above, define $\text{Good} = \{j \in [\kappa] : M_{i,j} = \mathsf{Aut}(m, K_{i,j})\}$. By a Chernoff bound, when IND_i is a random subset of $[\kappa]$ of size $\kappa/2$, the probability with which $\text{Good} = \text{IND}_i$ is negligible in κ. This implies the assumption.

Case: $\mathbf{S} \notin \mathcal{I}_0$ and $\mathbf{M} \in \mathcal{I}_k$. In this scenario also, the real and ideal executions are identical but with exactly one distinction: the output of the honest parties in the ideal execution is $\perp$ whenever the emulated honest receivers' output is not equal to the sender's message m.

We will prove that the output of the emulated honest parties belong to the set $\{m, \perp\}$. Since the emulation is identical to the real execution in the presence of $\mathcal{A}$, we will prove the statement in this setting.

Consider the event where all honest parties in $\mathcal{L}_{k'}$ has set $\texttt{success} = 0$. In this case, the output of all honest parties is m.

Next, consider the event $\texttt{success} = 1$. Observe that, since $\mathbf{S}$ is honest, for each honest $P_i \in \mathcal{L}_{k+1}$ and $j \in \text{IND}_i$, the values of $\bar{K}_{i,j}$ and $\tilde{K}_{i,j}$ stored by every honest party in $\mathcal{L}_{k+4}$ are identical. Hence, $i \notin \texttt{votes}$ (according to all honest parties in $\mathcal{L}_{k+4}$). Consequently, all parties in $\mathcal{L}_{k''}$ receive $\texttt{votes}$ such that $\texttt{votes} \subseteq \mathcal{I}_{k+1}$. Let $\tilde{m}$ and $\{\tilde{M}_{i,j}\}_{i \notin \texttt{votes}, j \in \text{IND}_i}$ be the values received by all honest parties in $\mathcal{L}_{k-2}$ from $\mathbf{M}$ (see steps 1 and 2 in $\mathcal{L}_{k-2}$). We claim that, in step 2. (b), no honest party in $\mathcal{L}_{k''}$ adds $i \in \mathcal{I}_{k+1}$ to $\texttt{votes}$ if $\tilde{m} \neq m$. This would directly imply security in this case.

The above claim is proved as follows: the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender with (purported) m as input, $\mathcal{L}_{k'}$ as auxilliary layer, and $\mathcal{L}_{k''-2}$ as receivers are independent of the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathcal{L}_{k''}$ as receivers. A similar claim holds the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender with $M_{i,j}$ as input, $\mathcal{L}_{k'}$ as auxilliary layer, and $\mathcal{L}_{k''-2}$ as receivers for each $i \notin \texttt{votes}, j \notin \text{IND}_j$. Hence, $\mathcal{A}$'s choice of $\tilde{m}$ and $\{\tilde{M}_{i,j}\}_{i \notin \texttt{votes}, j \in \text{IND}_i}$ is independent of $\{K_{i,j}\}_{i \notin \texttt{votes}, j \notin \text{IND}_i}$ *conditioned on* the values m and $\{M_{i,j}\}$ received from the honest sender. Hence, by unforgeability of MAC, $\Pr[\exists i \notin \texttt{votes}, j \in \text{IND}_i : \mathsf{Vfy}(\tilde{M}_{i,j}, \tilde{m}, K_{i,j}) = 1$ is negligible in κ by a union bound. In other words, provided an adversary who succeeds with a non-negligible probability, we can break the unforgeability condition of MAC with non-negligible advantage. This concludes the proof.

Case: $\mathbf{S} \notin \mathcal{I}_0$ and $\mathbf{M} \notin \mathcal{I}_k$. We first observe that, when $\mathbf{S}$ and $\mathbf{M}$ are honest, all honest parties in $\mathcal{L}_{k+6}$, computes $\texttt{success} = 1$. Consequently, $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{S}$ as sender and $\mathcal{L}_{k+6}$

as auxiliary layer is not revealed to receivers in $\mathcal{L}_{k'}$. Furthermore, for any $i \notin \texttt{votes}$ and $j \notin \text{IND}_i$, $(m, M_{i,j})$ revealed in $\mathcal{L}_{k''-2}$ satisfies the MAC check with respect to $K_{i,j}$ which is chosen uniformly. It can be verified that adversary's view and honest receivers' outputs are entirely determined by these random variables. The simulator arranges for the same correlation between these random variables, proving the equivalence. □

6.4 Generalizing Π_{ITSig} to implement $\mathcal{F}_{\mathsf{ITSig}}$

We now discuss the modifications that are needed to adapt the above protocol and proof to handle a message vector from the sender, support multiple auxiliary layers, and choose among multiple receiver layers.

Handling vector messages. First, we handle the the case where sender's message is a vector instead of a scalar, while still considering single auxiliary and receiving layers. We exploit the $\mathcal{D}$-linearity of MACs as described in Definition 3 (this basically just means that linearity holds as long as keys are sampled from an appropriate distribution). To sign a message vector $\mathbf{m} = (m_1, \ldots, m_\ell) \in M^\ell$, sender samples key vectors $\{\mathbf{K}_{i,j}\}_{i\in[n],j\in[\kappa]}$, where, for each i, j,

$$\mathbf{K}_{i,j} = (K^1_{i,j}, \ldots, K^\ell_{i,j}) \leftarrow_{\mathcal{D}} \mathcal{K}^\ell, \tag{36}$$

and computes the MAC vector

$$\mathbf{M}_{i,j} = (M^1_{i,j}, \ldots, M^\ell_{i,j}) = \left(\mathsf{Aut}(m_1, K^1_{i,j}), \ldots, \mathsf{Aut}(m_\ell, K^\ell_{i,j}\right), \tag{37}$$

for each i, j. The protocol proceeds as described above with the following differences:

1. The public MAC check of $(m, M_{i,j})$ reported by $\mathbf{M}$ using the key $\bar{K}_{i,j}$ reported by $\mathbf{S}$ for $i \in [n], j \in \text{IND}_i$ that occurs in step 1 of $\mathcal{L}_{k+6}$ is replaced by MAC check of each $(m_l, M^l_{i,j})$ reported by $\mathbf{M}$ using the key $\bar{K}^l_{i,j}$ reported by $\mathbf{S}$ for $i \in [n], j \in \text{IND}_i$ and $l \in [\ell]$. Success flag is set to 0 is any of these checks fail.

2. The parties in $\mathcal{L}_{k'}$ broadcasts $(m_1, \ldots, m_\ell)$ if $\texttt{success} = 0$ in step 1. Whereas, if $\texttt{success} = 0$, they reveal $m = L(m_1, \ldots, m_\ell)$ in step 2(a), and $M_{i,j} = L(M^1_{i,j}, \ldots, M^\ell_{i,j})$ for each $i \notin \texttt{votes}, j \notin \text{IND}_i$ to $\mathcal{L}_{k''-2}$ using $\mathcal{F}_{\mathsf{FutureBC}}$, where L is the linear functional that all honest parties in $\mathcal{L}_{k'}$ agree on.

3. Finally, parties in $\mathcal{L}_{k''}$ verify the purported values of of message m and MAC $M_{i,j}$ using $L(K^1_{i,j}, \ldots, K^\ell_{i,j})$ for each $i \notin \texttt{votes}, j \notin \text{IND}_i$

When the sender and intermediary are honest, only $L(m_1, \ldots, m_\ell)$ is revealed to the receivers. In this case, $\texttt{success}$ flag is always set to 1. Further, when the sender is honest, the keys distributed by the sender to honest parties in $\mathcal{L}_{k'+1}$ are unknown to the adversary. Since MAC is $\mathcal{D}$-linear, having learned the $\left(m, \{M^l_{i,j}\}_{l\in[\ell]}\right)$, an adversary corrupting the intermediary cannot choose $m' \neq L(m_1, \ldots, m_\ell)$ and $M'_{i,j}$ such that $\mathsf{Vfy}\left(m', M'_{i,j}, L(K^1_{i,j}, \ldots, K^\ell_{i,j})\right) = 1$ for any i such that $P^{k+1}_i \in \mathcal{I}_{k+1}$ since the keys are unknown to the adversary. As a consequence, when the sender is honest and intermediary is corrupt, the receivers either receive the right linear combination of the message vector or they abort.

Handling multiple auxiliary layers and receiver layers. In order to realize multiple auxiliary layers $\mathcal{L}_{k_1}, \ldots, \mathcal{L}_{k_w}$, we replicate the state held by $\mathcal{L}_{k'}$ in the above protocol on all the layers in our general version. This is achieved in a straightforward manner: when $\texttt{success} = 0$, $\mathcal{L}_{k+6}$ broadcasts the message vector from the sender, hence this value is available to all auxiliary layers which are necessarily downstream from this layer. To handle the case where $\texttt{success} = 1$, the intermediary sets up $\mathcal{F}_{\mathsf{FutureBC}}$ of $\mathbf{m}$ and $\mathbf{M}_{i,j}$ for each $i \in [n], j \in [\kappa]$, with $\mathcal{L}_{k_i}$ as auxiliary layer, for each $i \in [w]$.

Next, to set up multiple receiver layers $\mathcal{L}_{r_1}, \ldots, \mathcal{L}_{r_w}$, we essentially set up the same state on $\mathcal{L}_{r_1}, \ldots, \mathcal{L}_{r_v}$ as we did in $\mathcal{L}_{k''}$ in our construction above. However, the similarity here necessarily means that each $\mathcal{L}_{r_i}$ has fresh keys that have been unused for MAC verification so far. This is arranged by having parties in $\mathcal{L}_{k'+4}$ forward fresh keys from $[\kappa] \setminus \mathrm{IND}_i$ to each receiver layer for each $i \notin \texttt{votes}$. For this, parties in $\mathcal{L}_{k'+4}$ send the first $\kappa/2v$ key vectors in $[\kappa] \setminus \mathrm{IND}_i$ to $\mathcal{L}_{r_1}$, the next $\kappa/2v$ key vectors to $\mathcal{L}_{r_2}$ and so on.

An auxiliary layer $\mathcal{L}_{k_i}$ can reveal $L(\mathbf{m})$ to $\mathcal{L}_{r_j}$, where $k_i \leq r_j - 4$, by revealing $L(\mathbf{m})$ and $L(\mathbf{M}_{l,l'})$ for all l, l' such that $\mathbf{K}_{i,j}$ has been supplied to $\mathcal{L}_{r_j}$. Note that, here $l \in [n] \setminus \texttt{votes}$ and l' ranges over the ith block of $\kappa/2v$ keys in $[\kappa] \setminus \mathrm{IND}_i$. Since, in each receiver layer, the MAC check being performed is using fresh keys, an adversary corrupting the intermediary fails to forge the wrong linear combination as long as the sender is honest by linear homomorphism of MAC. If the sender's message is not made public due to $\texttt{success}$ set to 0, to each receiver layer, only the required linear combination of the message and MAC vectors are revealed; this ensures security.

Handling individual receivers. The functionality allows revealing the message to all parties in a receiver layer or to an individual parties in the layer specified by the auxiliary layer. As mentioned previously, it is easier to reveal the secret to a single party than to the whole layer. This is because, future broadcast to a single receiver does not leak the message to the adversary at an earlier layer (See description of $\mathcal{F}_{\mathsf{ITSig}}$). Consequently, both the keys and authenticated message can be directly revealed to the receiver bypassing the need for the early fixing of the authenticated message 2 layers previous to the target layer. These modifications yield the following Lemma, whose proof follows the same outline as the proof of Lemma 8.

Lemma 9. *If $t < n/2$ and assuming a linear $(\mathcal{D}, t, \mathsf{negl}(\kappa))$-secure* MAC*, there exists a protocol Π_{ITSig} which realizes functionality $\mathcal{F}_{\mathsf{ITSig}}$ with $(\mathsf{negl}(\kappa), t)$-statistical security in the $(\mathcal{F}_{\mathsf{FutureMsg}}, \mathcal{F}_{\mathsf{FutureBC}})$-hybrid model for a security parameter κ.*

7 Distributed Commitment

7.1 Distributed Commitment Functionality

We describe a distributed commitment functionality that allows parties to commit to values that can then be opened towards future layers. There might be multiple intermediate layers with the right to open a value. The functionality also allows for linear combinations of values to be opened. If the dealer was corrupted by the adversary at the time of commitment, then when an open request is submitted, the adversary is given the option to open the $\perp$ value (analogously as what happens with a traditional cryptographic commitment, where the dealer can always decide not to open a value). Notice the main differences between $\mathcal{F}_{\mathsf{DistCommit}}$ and $\mathcal{F}_{\mathsf{FutureBC}}$:

1. In $\mathcal{F}_{\mathsf{DistCommit}}$, even when the committer $\mathbf{C}$ is corrupted, then the adversary $\mathcal{A}$ must decide its inputs after some fixed number of layers, unlike in $\mathcal{F}^{k,k'}_{\mathsf{FutureBC}}$ where $\mathcal{A}$ can choose the value of a corrupt $\mathbf{S}$ until the very last moment. In other words, the adversary is *committed* to its inputs in $\mathcal{F}_{\mathsf{DistCommit}}$.

2. Functionality $\mathcal{F}_{\mathsf{DistCommit}}$ provides the ability for *multiple* layers to open the *same* commitments. While this can be achieved by $\mathcal{F}_{\mathsf{FutureBC}}$ when $\mathbf{S}$ is honest, in $\mathcal{F}_{\mathsf{DistCommit}}$ one has the guarantee that, even when $\mathbf{C}$ is corrupt, if two different layers open the *same* linear combination of commitments, in both cases the opened value will be the *same* (that is, if in both cases the opened value is not $\perp$).

Linear Distributed Commitment Functionality $\mathcal{F}^{k_1,\ldots,k_w}_{\mathsf{DistCommit}}$

Public Parameters. Committer $\mathbf{C} \in \mathcal{L}_0$. Auxiliary layers $\mathcal{L}_{k_i}$ $i \in [w]$ deciding which messages are opened. The domain M of the messages from $\mathbf{C}$. The maximum number of messages ℓ to be committed by $\mathbf{C}$.

Secret Inputs.

– From $\mathbf{C}$ messages $(m_1, \ldots, m_\ell) \in M$ to be committed.

– From each $P^{k_i}_j$ $i \in [w]$ message (L, r):

  - $L : M^\ell \to M$ the linear combination of $\mathbf{C}$'s messages to compute.
  - $r \in \mathcal{L}_{k'} \cup \{\mathcal{L}_{k'}\}$ is the intended recipient of this linear combination: either some specific party in $\mathcal{L}_{k'}$ or all the parties in layer $\mathcal{L}_{k'}$.

Layer $\mathcal{L}_0$: If $\mathbf{C} \notin \mathcal{I}_0$ receive messages $(m_1, \ldots, m_\ell)$ from $\mathbf{C}$.

Layer $\mathcal{L}_{k_i}$ for $i \in [w]$:

- If $\mathbf{C} \in \mathcal{I}_0$ receive from the (ideal) adversary messages $(m_1, \ldots, m_\ell)$ that $\mathbf{C}$ wants to commit.
- For each honest $P^{k_i}_i \in \mathcal{L}_{k_i} \setminus \mathcal{I}_{k_i}$, receive *the same* input (L, r).

Layer $\mathcal{L}_{k'-3}$: If $r = \mathcal{L}_{k'}$, then forward $(L, L(m_1, \ldots, m_\ell))$ to the (ideal) adversary.

Layer $\mathcal{L}_{k'}$:

- If $r = P^{k'}_s \in \mathcal{I}_{k'}$, then forward $\big(L, L(m_1, \ldots, m_\ell)\big)$ to the (ideal) adversary.

- If $\mathbf{C} \in \mathcal{I}_0$ receive from the (ideal) adversary boolean $\mathsf{open} \in \{0, 1\}$.
- If $\mathbf{C} \in \mathcal{I}_0$ and $\mathsf{open} = 1$, or if $\mathbf{C} \notin \mathcal{I}_0$:
    - If $r = P_s^{k'}$, send $L(m_1, \ldots, m_\ell)$ to $P_s^{k'}$.[a]
    - If $r = \mathcal{L}_{k'}$, send $L(m_1, \ldots, m_\ell)$ to all parties in $\mathcal{L}_{k'}$.
- If $\mathbf{C} \in \mathcal{I}_0$ and $\mathsf{open} = 0$:
    - If $r = P_s^{k'}$, send $\perp$ to $P_s^{k'}$.
    - If $r = \mathcal{L}_{k'}$, send $\perp$ to all parties in $\mathcal{L}_{k'}$.

[a]The projection map onto a component of $(m_1, \ldots, m_\ell)$ is linear, so that parties can decide to reconstruct exactly one of $\mathbf{C}$'s inputs.

7.2 Distributed Commitment Protocol

The distributed commitment protocol we present takes full advantage of the guarantees provided by $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$ from Section 6. To commit to value m, because no single intermediary can be trusted (they could be corrupted), a party creates a (t, n)-shamir sharing of m and then invokes $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$ with a different intermediary (in the same layer) for each share. Intuitively, only the shares corresponding to corrupted intermediaries (at most t) are leaked to the adversary. This is not a problem thanks to the t-privacy of the secret sharing scheme. Furthermore, even a dishonest dealer, or *committer*, is committed to the shares entrusted to honest intermediaries. Since the latter are at least $t+1$, they determine a unique polynomial and the dealer is now committed to the unique value identified by the honest intermediaries' shares.

Clearly, since $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$ provides no guarantees when both the dealer and an intermediary are corrupt, the reconstruction of the sharing might still fail, as the shares corresponding to dishonest intermediaries can be fixed arbitrarily by the adversary. However, if the reconstruction succeeds, the output of the reconstruction will be the unique value defined by the shares of honest intermediaries, providing the commitment property of our construction. The security of the protocol below, captured in Lemma 10, is proven in Section 7.3.

Linear Distributed Commitment Protocol $\Pi_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$

Public Parameters. Committer $\mathbf{C} \in \mathcal{L}_0$. Auxiliary layers $\mathcal{L}_{k_i}$ for $i \in [w]$ deciding which messages are opened. The domain M of the messages from $\mathbf{C}$. The maximum number of messages ℓ to be committed by $\mathbf{C}$. Latest layer $\mathcal{L}'_k$ onto which messages can be opened.

Secret Inputs. From $\mathbf{C}$ messages $(m_1, \ldots, m_\ell) \in M$.

Resources. Functionality $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$.

Layer $\mathcal{L}_0$: The committer **C** does:

- Sample a polynomial $f_i(x)$ of degree at most t such that $f_i(0) = m_i$ and do $(m_{i,1}, \dots, m_{i,n}) \leftarrow (f_i(1), \dots, f_i(n))$ for all $i \in [\ell]$.
- Input $(m_{1,j} \dots, m_{\ell,j})$ to $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\dots,k_w}[j]$[a] for $j \in [n]$ with intermediary $P_j^k \in \mathcal{L}_k$ for all $j \in [n]$.

<u>**Revealing $L(m_1, \dots, m_\ell)$ to $r \in \mathcal{L}_{k'} \cup \{\mathcal{L}_{k'}\}$**</u>**:**

Layer $\mathcal{L}_{k_i}$ for any $i \in [w]$: Each $P_s^{k_i}$ inputs (L, r) to $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\dots,k_w}[j]$ for all $j \in [n]$.

Layer $\mathcal{L}_{k'}$: Party $r = P_s^{k'}$ (or all parties in $\mathcal{L}_{k'}$ if $r = \mathcal{L}_{k'}$) does:

- Receive value l_j from $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\dots,k_w}[j]$ for all j.
- Interpolate a polynomial $\widehat{f}(x)$ through any of the $t+1$ points (i, l_i) for $l_i \neq bot$ and compute $\widehat{l} = \widehat{f}(x)$ and output l.

[a]This notation is used to identify the n distinct parallel instances of $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\dots,k_w}[j]$ for $j \in [n]$.

7.3 Distributed Commitment Security

Lemma 10. *If $t < n/2$ then the (n, t, k')-layered protocol $\Pi_{\mathsf{DistCommit}}^{k_1,\dots,k_w}$ realizes $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\dots,k_w}$ with $(0, t)$-statistical security in the $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\dots,k_w}$-hybrid model.*

Proof. For any adversary $\mathcal{A}$ we describe a simulator $\sigma(\mathcal{A})$ such that the joint distribution of the outputs of the adversary and honest parties in the real world (the adversary $\mathcal{A}$ interacting with protocol $\Pi_{\mathsf{DistCommit}}^{k_1,\dots,k_w}$ and functionality $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\dots,k_w}$) is identically distributed to that in the ideal world (the simulator $\sigma(\mathcal{A})$ interacting with functionality $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\dots,k_w}$).Given any adversary $\mathcal{A}$ interacting with protocol $\Pi_{\mathsf{DistCommit}}^{k_1,\dots,k_w}$ we describe the simulator σ below.

Simulator $\sigma(\mathcal{A})$ for Protocol $\Pi_{\mathsf{DistCommit}}^{k_1,\dots,k_w}$

The simulator $\sigma(\mathcal{A})$ emulates all honest parties and functionality $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\dots,k_w}$.

Layer $\mathcal{L}_0$: If $\mathbf{C} \notin \mathcal{I}_0$ is honest do:

- Compute a shamir sharing $(m_{i,1}, \dots, m_{i,n})$ of 0 for all $i \in [\ell]$.

Layer $\mathcal{L}_k$: If $\mathbf{C} \notin \mathcal{I}_0$ is honest, for all corrupt $P_j^k \in \mathcal{I}_k$ send $(m_{1,j}, \dots, m_{\ell,j})$ to $\mathcal{A}$ on behalf of the simulated functionality $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\dots,k_w}[j]$ for all $j \in [n]$.

Layer $\mathcal{L}_{k_1}$: If $\mathbf{C} \in \mathcal{I}_0$ is corrupt do:

- If $P_j^k \notin \mathcal{I}_k$ is honest receive messages $(m_{1,j}, \ldots, m_{\ell,j})$ from $\mathcal{A}$ on behalf of the simulated functionality $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}[j]$ for all $j \in [n]$.
- Interpolate m_i through $(m_{i,1}, \ldots, m_{i,n})$ for all $i \notin \mathcal{I}_k$, if the values don't lie on a polynomial of degree at most t set $m_i = \perp$.
- Input $(m_1, \ldots, m_\ell)$ to $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$.

Layer $\mathcal{L}_{k'-3}$: If $\mathbf{C} \notin \mathcal{I}_0$ is honest do:

- Receive $(L, L(m_1, \ldots, m_\ell))$ from $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$.
- For all $P_j^k \in \mathcal{I}_k$ compute $l_j = L\left(m_{1,j}, \ldots, m_{\ell,j}\right)$.
- Patch the simulated honest states to a valid sharing of $L(m_1, \ldots, m_\ell)$

$$\left((l_j)_{j \in \mathcal{I}_k}, \left(\widehat{l_j}\right)_{j \notin \mathcal{I}_k} \right). \tag{38}$$

- For each honest $P_j^k \notin \mathcal{I}_k$ forward l_j to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}[j]$.

Layer $\mathcal{L}_{k'}$:

- If $\mathbf{C}$ is honest, for each corrupt $P_j^k \in \mathcal{I}_k$ receive boolean $\mathsf{reveal}_j \in \{0,1\}$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}[j]$. If $\mathsf{reveal}_j = 0$ set $\widehat{l_j} \leftarrow \perp$, otherwise set $\widehat{l_j} \leftarrow l_j$ that was computed before.
- If $\mathbf{C}$ is corrupt, for each corrupt $P_j^k \in \mathcal{I}_k$ receive value $\widehat{l_j}$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}[j]$.
- If $\left(\left(\widehat{l_j}\right)_{j \in \mathcal{I}_k}, \left(\widehat{l_j}\right)_{j \notin \mathcal{I}_k} \right)$ lie on a polynomial of degree t then set $\mathsf{open} \leftarrow 1$, otherwise set $\mathsf{open} \leftarrow 0$.
- Input boolean open to $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$.
- Set the output of $\sigma(\mathcal{A})$ to the output of $\mathcal{A}$.

Let's first argue about the view of $\mathcal{A}$ in the real versus the ideal world. We consider the cases where $\mathbf{C}$ is honest or corrupt separately.Below, Sh and Rec denote Shamir Sharing and reconstruction.

Adversary View, Corrupt Committer C. In this case, by inspection of the protocol and the simulation we conclude that the view of the adversary in the real and ideal world is trivially identically distributed, as $\mathbf{C}$ is the only party with secret inputs and all values sent to $\mathcal{A}$ by $\sigma(\mathcal{A})$ are computing according to the protocol, as in the real world.

Adversary View, Honest Committer C. In this case, in $\mathcal{L}_k$ the adversary $\mathcal{A}$ learns shares $m_{i,j}$ for all $i \in [\ell]$ and $P_j^k \in \mathcal{I}_k$. In the ideal world, these shares are sampled according to $(m_{i,1}, \ldots, m_{i,n}) \leftarrow \mathsf{Sh}(0, r_i)$, while in the real world they are sampled according to $(m_{i,1}, \ldots, m_{i,n}) \leftarrow \mathsf{Sh}(m_i, r_i)$, where m_i for $i \in [\ell]$ are the honest $\mathbf{C}$'s inputs. However, thanks to the t-privacy of the secret sharing scheme, any subset of up to t shares are identically distributed regardless of the secret. Because $|\mathcal{I}_k| \leq t$ it follows that in these two scenarios the view of the adversary is identically distributed. Similarly, in layer $\mathcal{L}_{k'-3}$ the adversary $\mathcal{A}$ learns values l_j for $P_j^k \notin \mathcal{I}_k$. Both in the real and in the ideal world, thanks to the correct patching of the secret sharing scheme, these values are valid shares of honestly sampled sharings of $L(m_1, \ldots, m_\ell)$, and are therefore identically distributed.

Next, we argue about the distribution of outputs of the honest parties. Again, we distinguish the two cases where $\mathbf{C}$ is honest or corrupt. We only argue about the case of private outputs, but the case of a single honest receiver is completely analogous.

Honest Outputs, Corrupt Committer C. In this setting, in the ideal world, the outputs of honest parties in layer $\mathcal{L}_{k'}$ are simply $L(m_1, \ldots, m_\ell)$, where values $m_1, \ldots, m_\ell$ are the values input by $\sigma(\mathcal{A})$ to $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$ in layer $\mathcal{L}_{k_1}$. These value are computed as $\mathsf{Rec}(L(\{m_{i,j}\}_{j \notin \mathcal{I}_k}))$ In the real world, the output of honest parties are computed by first checking that all shares (for all $j \in [n]$) lie on a polynomial of degree at most t and then choosing any $t+1$ to reconstruct the secret. However, the $t+1$ shares corresponding to honest P_j^k (which are the same in the real and ideal world because of $\mathcal{F}_{\mathsf{ITSig}}$) determine a unique polynomial of degree at most t, so the outputs in the real and ideal world are the same.

Honest Outputs, Honest Committer C. In this case, in the ideal world the output of honest parties is simply $L(m_1, \ldots, m_\ell)$, where values m_i for $i \in [\ell]$ are the inputs of the honest $\mathbf{C}$. In the real world, the output of honest parties is computed as $\mathsf{Rec}(l_1, \ldots, l_n)$, where value l_j for $j \in [n]$ is computed by $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$ as $L(m_{j,1}, \ldots, m_{j,n})$, and these are the values input to the functionality by P_j^k. For $P_j^k \in \mathcal{I}_0$ these values can be set to $\perp$ by $\mathcal{A}$, but for $P_j^k \notin \mathcal{I}_0$, the values $m_{i,j}$ are sampled honestly from $\mathbf{C}$ as $(m_{i,1}, \ldots, m_{i,n}) \leftarrow \mathsf{Sh}(m_i, r_i)$. Since $|\mathcal{I}_k| \leq t$, these values determine a unique polynomial of degree t. $\square$

8 Parallel Linear VSS

8.1 Parallel Linear VSS Functionality

In this section we describe our parallel VSS functionality $\mathcal{F}_{\mathsf{VSS}}^{k_1,\ldots,k_w}$. The functionality can be thought of as a strong distributed commitment functionality: parties in an initial layer $\mathcal{L}_0$ can input (commit to) values, and then later parties (in layers $k_1, \ldots, k_w$) can decide to perform linear operations on the values and reveal them as long as the majority of parties in the layer agree (from which the distributed nature of the commitment). Parties are *strongly committed* to the values they input, in the sense that they cannot abort the opening of these values, or linear combinations of them, at a later time: this is the first big difference between this functionality and $\mathcal{F}_{\mathsf{DistCommit}}$. The second major difference is that parties in layers $k_1, \ldots, k_w$ can perform linear operations on values committed by *different* parties (in the functionality, this is modeled by the linear function L). This stronger linearity property is crucial to perform secure addition and multiplication.

Parallel Linear VSS Functionality $\mathcal{F}_{\mathsf{VSS}}^{k_1,\ldots,k_w}$

Public Parameters. Committers $\mathbf{C}_1, \ldots, \mathbf{C}_n \in \mathcal{L}_0$. Auxiliary layers $\mathcal{L}_{k_i}$ for $i \in [w]$ deciding which messages are opened. The domain M of the messages from $\mathbf{C}_i$ for all $i \in [n]$. The maximum number of messages ℓ to be committed by each committer.

Secret Inputs.

– From each honest $\mathbf{C}_i$ messages $(m_{1,i}, \ldots, m_{\ell,i}) \in M$ to be committed.

– From each $P_j^{k_i}$ $i \in [w]$ message $(L, L_1, \ldots, L_n; r)$:

  - $L_i : M^\ell \to M$ the linear combination of $\mathbf{C}_i's$ inputs to compute.
  - $L : M^n \to M$ the linear combination of these linear combinations to compute.
  - $r \in \mathcal{L}_{k'} \cup \{\mathcal{L}_{k'}\}$ is the intended recipient of this linear combination: either some specific party in $\mathcal{L}_{k'}$ or all the parties in layer $\mathcal{L}_{k'}$.

Layer $\mathcal{L}_0$: For each $\mathbf{C}_i \notin \mathcal{I}_0$ receive messages $(m_{1,i}, \ldots, m_{\ell,i})$ from $\mathbf{C}_i$.

Layer $\mathcal{L}_{k_i}$ for $i \in [w]$:

- For each $\mathbf{C}_i \in \mathcal{I}_0$ receive from the (ideal) adversary messages $(m_{1,i}, \ldots, m_{\ell,i})$ that $\mathbf{C}_i$ wants to commit.
- For each honest $P_i^k \in \mathcal{L}_k \setminus \mathcal{I}_k$, receive *the same* input $(L, L_1, \ldots, L_n; r)$.

Layer $\mathcal{L}_{k'-3}$:

- Let $l = L\big(L_1(m_{1,1}, \ldots, m_{\ell,1}), \ldots, L(m_{1,n}, \ldots, m_{\ell,n}))\big)$.

- If $r = \mathcal{L}_{k'}$, then forward $(L, L_1, \ldots, L_n; l)$ to the (ideal) adversary.

Layer $\mathcal{L}_{k'}$:

- If $r = P_s^{k'}$, send the value l to $P_s^{k'}$.
- If $r = \mathcal{L}_{k'}$, send the value l to all parties in $\mathcal{L}_{k'}$.

8.2 Linear VSS Protocol

Starting from the distributed commitment functionality $\mathcal{F}_{\mathsf{DistCommit}}$ we construct a protocol that realizes $\mathcal{F}_{\mathsf{VSS}}$ with *perfect* security.

The task is to prevent a *corrupt* dealer from disrupting the opening of their commitment at a later time. A basic idea is to ask the dealer to robustly secret share the input s and send each share to a distinct party in a following layer. Each party can then commit to their share. This simple approach fails because the secret sharing provides no guarantees when the dealer is corrupted: the adversary can selectively abort the reconstruction by preventing the opening of different subsets of corrupted commitments.

An even bigger problem is that even with an honest dealer, corrupted parties can commit to arbitrary values. To tackle both these problems at once, we ask the dealer to commit to the randomness used in the sharing (in our case, a polynomial) and prove in ZK that that they are providing valid shares (with respect to this randomness) to the auxiliary layer. Then, parties in the auxiliary layer prove in ZK that they are committing to the values received from the dealer. Since we cannot rely on public randomness for these distributed ZK-proofs (we are using this VSS protocol to implement our random beacon later) we resort to techniques based on bi-variate polynomials and leverage the honest majority in each layer.

The dealer produces a two-dimensional polynomial sharing of their input, and sends each share (now a univariate vertical projection of the bi-varate polynomial) to a distinct party in an auxiliary layer. The dealer also commits to their bi-variate polynomial. Each of the parties in the auxiliary layer now commits to the polynomial received from the dealer. The opening of commitments to the dealer's polynomial can fail if the dealer is corrupted, but the projections committed by honest parties in the auxiliary layer will open correctly, even if those by dishonest parties might be inconsistent with the dealer's polynomial. To ensure consistency of all projections with the dealer's polynomial, every horizontal projection of the dealer's polynomial and the corresponding cross points with the vertical projections are opened towards distinct parties in another layer. If any inconsistency is detected, the conflict is then publicly resolved. Privacy is not an issue as there are no inconsistencies between an honest dealer's polynomial and honest parties' projections.

Linear VSS Protocol $\Pi_{\mathsf{VSS}}^{k_1,\ldots,k_w}$

Public Parameters. Dealer $\mathbf{D} \in \mathcal{L}_0$. Layers $\mathcal{L}_{k_1}, \ldots, \mathcal{L}_{k_w}$ receiving sharing states. Domain $\mathcal{S}$ of the secret. The domain $\mathbf{S}$ of each share. Finite field $\mathbb{F}$.

Secret Inputs. From $\mathbf{D}$ the secret s to share.

Resources. Functionality $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$ (as the functionality is only invoked with this set of parameters throughout the protocol, in the description below we omit the parameters for legibility).

Layer $\mathcal{L}_0$: The dealer $\mathbf{D}$ does:

- Sample $f_{k,\ell} \leftarrow_{\$} \mathbb{F}$ for $k, \ell \in \{0,\ldots,t\}$ such that $(k,\ell) \neq (0,0)$.
- Let
$$(x,y) = \sum_{k,\ell=0}^{t} f_{k,\ell} x^k y^\ell \tag{39}$$
  where $f_{0,0} = s$.
- Let $h_i(x) = F(x,i) = \sum_{k=0}^{t} h_{i,k} x^k$.
- Let $v_i(y) = F(i,y) = \sum_{k=0}^{t} v_{i,k} y^k$.
- For all $k, \ell \in [0,t]$ input $f_{k,\ell}$ to $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$.[a]
- Send $(v_{i,0},\ldots,v_{i,t})$ to P_i^1 via bilateral secure channels.

Layer $\mathcal{L}_1$: Each P_i^1 does:

- If $v_{i,k}$ was not received for some $k \in \{0,\ldots,t\}$ set $v_{i,k}$ to 0.
- Input $v_{i,k}$ to $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for all $k \in \{0,\ldots,t\}$.

Layer $\mathcal{L}_{k_1}$: Each $P_s^{k_1}$ does:

- Input $\left(L = (j^0,\ldots,j^t), r = P_j^{k_2}\right)$ to $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for $k \in \{0,\ldots,t\}$.[b]
- Input $\left(L = (i^0 j^0,\ldots,i^t j^t), r = \mathcal{L}_{k_4}\right)$ to $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$ for $i \in [n]$.

Layer $\mathcal{L}_{k_2}$: Each $P_j^{k_2}$ does:

- Receive value $\widehat{v_i(j)}$ from $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for all $i \in [n]$.
- Receive values $\widehat{h}_j(i)$ from $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$ for all $i \in [n]$.
- If $\widehat{v_i(j)} = \bot$, or $\widehat{h_j(i)} = \bot$, or $\widehat{v_i(j)} \neq \widehat{h_j(i)}$, then broadcast $\mathsf{complain}_{i,j}$.

Layer $\mathcal{L}_{k_3}$: Each $P_s^{k_3}$ does:

- For all $i, j \in [n]$, if $\mathsf{complain}_{i,j}$ was broadcast by $P_j^{k_2}$ do:
  - Input $\big(L = (j^0, \ldots, j^t), r = \mathcal{L}_{k_4}\big)$ to $\mathcal{F}_{\mathsf{DistCommit}}[i]$.[c]
  - Input $\big(L = (i^0 j^0, \ldots, i^t j^t), r = \mathcal{L}_{k_4}\big)$ to $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$.[d]

Layer $\mathcal{L}_{k_4}$: Each $P_s^{k_4}$ does:

- For all $i, j \in [n]$, if $\mathsf{complain}_{i,j}$ was broadcast by $P_j^{k_2}$, do:
  - Receive value $\overline{v_i(j)}$ from $\mathcal{F}_{\mathsf{DistCommit}}[i]$.
  - Receive value $\overline{h_j(i)}$ from $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$.
  - If $\overline{h_j(i)} = \perp$ the dealer $\mathbf{D}$ is disqualified.
  - If $\overline{v_i(j)} = \perp$ or $\overline{v_i(j)} \neq \overline{h_j(i)}$, then add index i to a set $\mathcal{I}$ (if $|\mathcal{I}| > t$ then $\mathbf{D}$ is disqualified) and for all $j \in [n]$ do:
    - Input $\big(L = (j^0, \ldots, j^t), r = \mathcal{L}_{k_5}\big)$ to $\mathcal{F}_{\mathsf{DistCommit}}[i]$.[e]
    - Input $\big(L = (i^0 j^0, \ldots, i^t j^t), r = \mathcal{L}_{k_4}\big)$ to $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$.[f]

Layer $\mathcal{L}_{k_5}$: Each $P_s^{k_5}$ does:

- Receive values $\overline{v_i(j)}$ for all $j \in [n]$ from $\mathcal{F}_{\mathsf{DistCommit}}[i]$. 2
- Receive values $\overline{h_j(i)}$ for all $j \in [n]$ from $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$.
- If $\overline{h_j(i)} = \perp$ for any $j \in [n]$, then the dealer $\mathbf{D}$ is disqualified.

<u>Sharing State VSS:</u>

Public State. Set $\mathcal{I}$ and each $i \in \mathcal{I}$ polynomial $v_{i,\mathbf{D}}(y)$ through values $\overline{h_j(i)}$ for all $j \in [n]$ revealed in layer $\mathcal{L}_{k_5}$.
Private State. For all $i \notin \mathcal{I}$ the state of functionality $\mathcal{F}_{\mathsf{DistCommit}}[i]$.

<u>Dealer With Multiple Inputs $(s_1, \ldots, s_\ell)$:</u>

We described the protocol in the case where $\mathbf{D}$ only has one input s to avoid a notational blow-up. However, it is easy to generalize the construction to the case where $\mathbf{D}$ has multiple inputs $s_1, \ldots, s_\ell$. The protocol is simply executed ℓ times in parallel, but using the *same* instances of $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for all $\iota \in [n]$ and $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$, which allow for an arbitrary number of inputs. This ensures homomorphism across the sender's inputs. Furthermore, in each parallel execution with input s_ι for $\iota \in [\ell]$ the set $\mathcal{I}^{(\iota)}$ might be different: we consider $\mathcal{I} = \bigcup_{\iota \in [\ell]} \mathcal{I}^{(\iota)}$ to be the union of all such sets. Again, if $|\mathcal{I}| > t$ then $\mathbf{D}$ is disqualified.

<u>Revealing $L(s_1, \ldots, s_\ell)$ to $r \in \mathcal{L}'_k \cup \{\mathcal{L}_{k'}\}$:</u>

Suppose $\mathbf{D}$ has inputs $s_1, \ldots, s_\ell$. If $\mathcal{D}$ was disqualified in any of the executions corresponding to any s_ι the honest party simply output 0.

Layer $\mathcal{L}_{k_r}$ for $r \geq 5$: Each $P_s^{k_i}$ does:

- For all $i \notin \mathcal{I}$ input (L', r) to $\mathcal{F}_{\mathsf{DistCommit}}[i]$, where L' is the linear function that comptutes $L(\widehat{v}_i^{(1)}(0), \ldots, \widehat{v}_i^{(n)}(0))$.

- For all $i \in \mathcal{I}$ input (π'_ι, r) to $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for all $\iota \in [\ell]$.[g]

Layer $\mathcal{L}_{k'}$: Each $P_s^{k'}$ does:

- Receive output l_i from $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for $i \notin \mathcal{I}$.

- Receive outputs $s_{i,\iota}$ from $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for $i \in \mathcal{I}$ and $\iota \in [\ell]$.

- Let $l_i = L(s_{1,i}, \ldots, s_{\ell,i})$ for $i \in \mathcal{I}$.

- Interpolate the unique polynomial $\widehat{f}(x)$ through any of $t+1$ among points $(i, l_i)_{i \in [n]}$ and output $\widehat{f}(0)$.

Multiple Dealers $\mathbf{D}_1, \ldots, \mathbf{D}_n$:

We described the protocol in the case where there is only one dealer to avoid a notational blow-up. However, when invoked in parallel by different dealers the protocol provides linearity among values dealt by *different* dealers. Consider parallel executions with dealer $\mathbf{D}_\iota$ with input s_ι for $\iota \in [n]$. Suppose party P_i^1 commits to polynomials $\widehat{v}_i^{(\iota)}(y)$ in execution with $\mathbf{D}_\iota$. Revealing a linear combination of values $L'(s_1, \ldots, s_n)$ can be done by revealing the corresponding linear combination of the polynomials $\widehat{v}_i^{(\iota)}(y)$. Some care must be put to ensure privacy. Indeed, in executions with different dealers the sets $\mathcal{I}^{(\iota)}$ could be different. Suppose that $P_i^1 \in \mathcal{I}^{(1)}$ but $P_i^1 \notin \mathcal{I}^{(\iota)}$ for all other $\iota \neq 1$. Then, we cannot simply reconstruct all polynomials $\widehat{v}^{(i)}(0)$ for $\iota \neq 1$ and compute L' on public information, as this would violate the privacy of honest dealer's values. Therefore, we first compute the projection of L' on $\widehat{v}^{(i)}(0)$ for $\iota \neq 1$, and then add $\widehat{v}^{(1)}(0)$ afterwards. Details follow.

Revealing $L(s_1, \ldots, s_n)$ to $r \in \mathcal{L}'_k \cup \{\mathcal{L}_{k'}\}$:

Suppose dealer $\mathbf{D}_\iota$ has input s_ι.

Layer $\mathcal{L}_{k_r}$ for $r \geq 5$: Each $P_s^{k_i}$ does:

- If $L = (a_1, \ldots, a_n)$ then let $L' = (\widetilde{a}_1, \ldots, \widetilde{a_n})$ where $\widetilde{a}_\iota \leftarrow a_\iota$ if $i \notin \mathcal{I}^{(\iota)}$, and $\widetilde{a}_\iota \leftarrow 0$ if $i \in \mathcal{I}^{(\iota)}$.

- For all $i \in [n]$ input (L', r) to $\mathcal{F}_{\mathsf{DistCommit}}[i]$.

Layer $\mathcal{L}_{k'}$: Each $P_s^{k'}$ does:

- Receive output l'_i from $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for $i \in [n]$.
- If $l'_i \neq \perp$ let $l_i = l'_i + \sum_{\iota \text{ such that } i \in \mathcal{I}^{(\iota)}} v^{(\iota)}_{i,\mathbf{D}_\iota}(0)$ for all $i \in [n]$.
- Interpolate the unique polynomial $\widehat{f}(x)$ through any of $t+1$ among points (i, l_i) and output $\widehat{f}(0)$.

[a]This notation is used to identify different parallel instances of $\mathcal{F}_{\mathsf{DistCommit}}$.
[b]This reveals $\widehat{v}_i(j)$ to $P_j^{k_2}$.
[c]This publicly reveal values $\widehat{v_i(j)}$ to all parties in $\mathcal{L}_{k_3}$.
[d]This publicly reveals $\widehat{F}(i,j) = \widehat{h}_j(i)$ to all parties in layer $\mathcal{L}_4$.
[e]This reveals values $\overline{h_j(i)}$ for all $j \in [n]$ to all parties in $\mathcal{L}_{k_5}$.
[f]This reveals values $\overline{v_i(j)}$ for all $j \in [n]$ to all parties in $\mathcal{L}_{k_5}$.
[g]We denote by π'_ι the linear projection map computing $\widehat{v}_i^{(\iota)}(0)$ for $\iota \in [\ell]$.

8.3 Linear VSS Security

Lemma 11. *If $\mathbf{D}$ is honest in an execution of $\Pi_{\mathsf{VSS}}^{k_1,\ldots,k_w}$ then $\mathcal{I} \subseteq \mathcal{I}_1$, and therefore also $|\mathcal{I}| \leq t$.*

Proof. If $\mathbf{D}$ is honest, then each honest party $P_i^1 \notin \mathcal{I}_1$ receives from $\mathbf{D}$ polynomial $v_i(y) = F(i, y)$. Therefore, in $\mathcal{L}_{k_4}$ it always holds that, if P_i^1 is honest, then

$$\overline{v_i(j)} = F(i,j) = \overline{h_j(i)}, \tag{40}$$

so that $i \notin \mathcal{I}$. Since there are at most t corrupted parties in $\mathcal{I}_1$, it also follows that $|\mathcal{I}| \leq t$. □

Lemma 12. *In an execution of $\Pi_{\mathsf{VSS}}^{k_1,\ldots,k_w}$ there exists a polynomial $\widehat{F}(x,y)$ such that 1) for $i \notin \mathcal{I}$ then party P_i^1's inputs to $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}[i]$ are the coefficients of $\widehat{F}(i,y)$, and 2) for $i \in \mathcal{I}$ then $v_{i,D}(y) = \widehat{F}(i,y)$. If $\mathbf{D}$ is honest then $\widehat{F}(x,y) = F(x,y)$.*

Proof. Let $\widehat{F}(x,y)$ denote the unique polynomial determined by dealer's inputs to functionality $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}[\mathbf{D}]$ for all $j \in [n]$. Furthermore let $\widehat{v}_i(y)$ denote the unique polynomials determined by party P_i^1's inputs to $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}[i]$ for all $i \in [n]$. We argue that for all $i \in [n]$ we have $v_{i,\mathbf{D}}(y) = \widehat{v}_i(j)$. We distinguish two cases:

Claim 1) Suppose that $\widehat{v}_i(y) \neq \widehat{F}(i,y)$. Since both these polynomials have degree at most t, they differ in at least $n - t \geq t + 1$ points. This means that at least one honest $P_j^{k_2} \notin \mathcal{I}_{k_2}$ would have broadcast a complaint $\mathsf{complain}_{i,j}$ which could not be answered correctly by the dealer in $\mathcal{L}_{k_4}$, which in turn would mean $i \in \mathcal{I}$, a contradiction.

Claim 2) This case is trivial because point $v_{i,\mathbf{D}}(j)$ is by definition the same as $\widehat{F}(i,j)$ (it is computed by $\mathcal{F}^{k_1,\dots,k_w}_{\mathsf{DistCommit}}[\mathbf{D}]$ in exactly the same way).

□

Now notice that at the time of reconstruction the values $l_i = \widehat{v}_i(0)$ of honest parties with $i \notin \mathcal{I}$ will not be $\perp$. Together with the polynomial $v_{i,\mathbf{D}}$ for $i \in \mathcal{I}$ there are at least $t+1$ values $l_i \neq \perp$. All the following properties follow trivially from this.

Lemma 13. . *Assume that $t < n/2$. The (n,t,k')-layered protocol $\Pi^{k_1,\dots,k_w}_{\mathsf{VSS}}$ realizes functionality $\mathcal{F}^{k_1,\dots,k_w}_{\mathsf{VSS}}$ with $(0,t)$-statistical security (i.e. perfect security) in the $\mathcal{F}^{k_1,\dots,k_w}_{\mathsf{DistCommit}}$-hybrid model.*

Proof. For any adversary $\mathcal{A}$ we describe a simulator $\sigma(\mathcal{A})$ such that the joint distribution of the outputs of the adversary and honest parties in the real world (the adversary $\mathcal{A}$ interacting with protocol $\Pi^{k_1,\dots,k_w}_{\mathsf{VSS}}$ and functionality $\mathcal{F}^{k_1,\dots,k_w}_{\mathsf{DistCommit}}$) is identical to that in the ideal world (the simulator $\sigma(\mathcal{A})$ interacting with functionality $\mathcal{F}^{k_1,\dots,k_w}_{\mathsf{VSS}}$). This shows that $\mathcal{F}^{k_1,\dots,k_w}_{\mathsf{VSS}}$ can be realized from $\mathcal{F}^{k_1,\dots,k_w}_{\mathsf{DistCommit}}$ with *perfect* security. First, given any adversary $\mathcal{A}$ interacting with protocol $\Pi^{k_1,\dots,k_w}_{\mathsf{VSS}}$ we describe the simulator σ. For ease of notation we only describe the simulator for the case of one dealer and one input, but the general case is analogous up to notational hurdles.

Simulator $\sigma(\mathcal{A})$ for Protocol $\Pi^{k_1,\dots,k_w}_{\mathsf{VSS}}$

The simulator $\sigma(\mathcal{A})$ executes the protocol on behalf of honest parties. We highlight how the simulator deals with the missing inputs of an honest dealer.

Layer $\mathcal{L}_0$: If $\mathbf{D} \notin \mathcal{I}_0$ do:

- Sample $f_{k,\ell} \leftarrow_{\$} \mathbb{F}$ for $k,\ell \in \{0,\dots,t\}$ such that $(k,\ell) \neq (0,0)$.
- Let $f_{0,0} \leftarrow 0$ (the simulator sets the dealer's input to 0).
- Let $F(x,y) = \sum_{k,\ell=0}^{t} f_{k,\ell} x^k y^\ell$.
- For each corrupt $P^1_i \in \mathcal{I}_1$ send $F(i,y)$ to P_j.

Layer $\mathcal{L}_1$:

- For each corrupt $P^1_i \in \mathcal{I}_1$ receive coefficients $\widehat{v}_{i,k}$ for $k \in \{0,\dots,t\}$ on behalf of $\mathcal{F}^{k_1,\dots,k_w}_{\mathsf{DistCommit}}[i]$.

Layer $\mathcal{L}_{k_1}$:

- For each $P^1_i \in \mathcal{I}_1$ receive from $\mathcal{A}$ coefficients $\widehat{h_{i,k}}$ for $k \in \{0,\dots,t\}$ on behalf of functionality $\mathcal{F}^{k_1,\dots,k_w}_{\mathsf{DistCommit}}[i]$.

- If $\mathbf{D} \in \mathcal{I}_0$ receive from $\mathcal{A}$ coefficients $\widehat{f}_{k,\ell}$ for $k, \ell \in \{0, \ldots, t\}$ that the dishonest dealer wants to commit, on behalf of functionality $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}[\mathbf{D}]$.
- If $\mathbf{D} \in \mathcal{I}_0$ and $\mathbf{D}$ was not disqualified in the simulation, input $f_{0,0}$ to $\mathcal{F}_{\mathsf{VSS}}^{k_1,\ldots,k_w}$. If $\mathbf{D}$ was disqualified in the simulation, input 0 to $\mathcal{F}_{\mathsf{VSS}}^{k_1,\ldots,k_w}$.

Then $\sigma(\mathcal{A})$ continues to emulate the protocol and functionalities $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}[i]$ and $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}[\mathbf{D}]$ on behalf of the honest parties (using the simulated states in the case $\mathbf{D} \in \mathcal{I}_0$).

Layer $k' - 3$:

- If $\mathbf{D} \notin \mathcal{I}_0$ receive from $\mathcal{F}_{\mathsf{VSS}}^{k_1,\ldots,k_w}$ the actual dealer's input s.
- Sample a uniform random polynomial $\widehat{F}(x, y)$ of degree at most t in each variable such that $\widehat{F}(i, y) = F(i, y)$ for all $i \in \mathcal{I}_1$ and $\widehat{F}(x, j) = F(x, j)$ for all $j \in \mathcal{I}_{k_2}$. This can be done efficiently by first interpolating the unique polynomial through $F(i, 0)$ for $i \in \mathcal{I}_1$ and s, let it be $h(x)$, and then interpolating the polynomial through $F(i, j)$ and $h(i)$ for $j \in \mathcal{I}_{k_2}$ for all $i \notin \mathcal{I}_0$. It is clear that all these polynomials are consistent and define a unique polynomial $\widehat{F}(x, y)$.
- For each $P_i^1 \notin \mathcal{I}_1$ send the coefficients of $\widehat{F}(i, y)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}[i]$.

Layer k':

- Set the output of $\sigma(\mathcal{A})$ to the output of $\mathcal{A}$.

We first start by arguing that the view of $\mathcal{A}$ in the real and ideal world is identically distributed. When the dealer $\mathbf{D}$ is corrupted, the simulator simply executes the protocol on behalf of the honest parties, so this case is trivial. When the dealer is honest, by inspection of the protocol and the simulation we conclude that the only difference in the view of the adversary in the two scenarios is that 1) in the ideal world $\mathcal{A}$ learns polynomials $F(i, y)$ for all $i \in \mathcal{I}_1$ and $F(x, j)$ for all $j \in \mathcal{I}_{k_2}$ respectively. These are projections of a polynomial $F(x, y)$ sampled uniform at random conditioned on that facts that the degree in each variable is at most t and that $F(0, 0) = 0$, while 2) in the real world the polynomials the adversary $\mathcal{A}$ learns are the projections of a polynomial $\widehat{F}(x, y)$ which is sampled uniformly at random conditioned on that facts that the degree in each variable is at most t and that $F(0, 0) = s$, where s is the actual input of the honest dealer. However, a subset of up to t vertical and horizontal projection is independent from the secret s. More formally, once these projections are fixed there exists exactly one polynomial for each possible secret s' with the same projections. Therefore, the view of the adversary is identically distributed in these two scenarios.

Next, we argue about the outputs of honest parties. We distinguish two cases. First suppose that $\mathbf{D}$ is honest. In the ideal world, the outputs of honest parties (who receive output) is simply the honest dealer's input s. In the real world, the output of the honest parties is computed by interpolating any the polynomial $\widehat{f}(x)$ through $t+1$ of the values $(i, \widehat{v}_i(0))$ for $i \notin \mathcal{I}$ and $(i, v_{i,\mathcal{D}}(0))$ for $i \in \mathcal{I}$ and outputting $\widehat{f}(0)$. By Lemma 11 the dealer is not disqualified, so that by Lemma 12 we can conclude $\widehat{f}(x) = F(x,0)$, so that $\widehat{f}(0) = s$. Now suppose that $\mathbf{D}$ is corrupted. The dealer is disqualified in the simulation if and only if it is disqualified in the real world, because all messages are computed by $\sigma(\mathcal{A})$ according to the protocol. In this case, in both the ideal and the real world the output of honest party is 0 (because in the ideal world $\sigma(\mathcal{A})$ sets the corrupted dealer's input to 0 in $\mathcal{F}_{\mathsf{VSS}}^{k_1,\ldots,k_w}$). If the dealer is not disqualified, then in the ideal world the output of honest parties is $\widehat{f}_{0,0}$, and in the real world by Lemma 12 the output of honest parties is also $\widehat{f}(x) = \widehat{F}(x,0)$, so that $\widehat{f}(0) = \widehat{f}_{0,0}$. □

9 Uniform Random Beacon

Our beacon functionality, when queried by all honest parties in a layer, samples uniform random field elements and provides this value to parties in a later layer.

9.1 Uniform Random Beacon Functionality

Beacon Functionality $\mathcal{F}_{\mathsf{Beacon}}^{k}$

Public Parameters. The output space $\mathcal{R}$ of values of $\mathcal{F}_{\mathsf{Beacon}}^{k}$. Layer $\mathcal{L}_0$ deciding whether or not to sample a value. Layer $\mathcal{L}_k$ receiving the sampled value.

Secret Inputs. From each $P_i^0 \in \mathcal{L}_0$ boolean value $\mathsf{sample}_j \in \{0,1\}$.

Layer $\mathcal{L}_0$:

- Receive *the same* input sample from all $P_i^0 \notin \mathcal{I}_0$.

Layer $\mathcal{L}_k$:

- If $\mathsf{sample} = 1$ sample $r \leftarrow_\$ \mathcal{R}$.
- Send r to all parties in $\mathcal{L}_k$.

9.2 Uniform Random Beacon Protocol and Security

Given our $\mathcal{F}_{\mathsf{VSS}}^{k}$ functionality, implementing $\mathcal{F}_{\mathsf{Beacon}}^{k}$ is straight-forward using a simple commit-and-open approach: each party in $\mathcal{L}_0$ commits to a uniform random value in $\mathcal{R}$ by inputting this value to $\mathcal{F}_{\mathsf{VSS}}^{k}$ and then parties in $\mathcal{L}_k$ ask $\mathcal{F}_{\mathsf{VSS}}^{k}$ to open the sum of all committed values. Clearly, this approach is quite wasteful because from $n-t$ uniform random values input by honest parties in $\mathcal{L}_k$ only one public random value is produced, but we choose it for its simplicity.

10 Circuit Evaluation

In this section, we explain how the layered components we developed so far can be used to securely evaluate a circuit C encoding a function f. Let $\mathbb{F}$ be a finite field, and let f be a function $f : \mathbb{F}^{\ell} \to \mathbb{F}^{\ell'}$. We denote by $\mathcal{F}_f$ the functionality that takes a set of inputs $(s_i)_{i \in \mathsf{inputs}_i}$ from all input clients $\mathcal{C}_i$ for $i \in [n]$ belonging to an initial layer $\mathcal{L}_0$, and delivers $f(s_1, \ldots, s_\ell)_{j \in \mathsf{outputs}_i}$ to the output clients $\mathcal{C}'_1, \ldots, \mathcal{C}'_n$.

Theorem 1. *Let $t < n/2$. If C is a circuit with depth d computing f, the $(n, t, O(d))$ layered protocol Π^C_{MPC} realizes functionality $\mathcal{F}_f$ with $(\mathsf{negl}(\kappa), t)$-statistical security in the $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{Beacon}})$-hybrid model.*

Protocol Invariant. Throughout the circuit evaluation, we maintain the following invariant: there is a layer of parties, say $\mathcal{L}_0$, holding a state encoding the input values a and b to every gate g in a certain level of the circuit C, and a layer $\mathcal{L}_k$ holding a state encoding the output $c = g(a, b)$ of g. It is clear that this invariant allows performing the computation of the whole circuit layer by layer. The state encoding an input value a to a gate g has three components:

1. A (t, n)-Shamir Sharing of a, where each party $P^0_i \in \mathcal{L}_0$ holds share a_i.
2. A (t, n)-Shamir Sharing of a random value r (which is used to securely evaluate multiplication gates).
3. Commitments (in the strong sense, produced using $\mathcal{F}_{\mathsf{VSS}}$) to the coefficients of the polynomial $f_a(x)$ used for the Shamir Sharing of a. Observe that, because of the linear properties of $\mathcal{F}_{\mathsf{VSS}}$, this means that parties also hold commitments to *each share* a_i.

The need for such a cumbersome state is somewhat inherent to the $t \geq n/3$ honest majority setting. Unlike in the $t < n/3$ case, where robust linearly homomorphic sharings have a simple description (typically a polynomial sharing, for example Shamir sharing), in the honest majortiy setting combining robustness and linearity is trickier. In particular, the robust sharing scheme from Section 3.3 does *not* provide linearity across different dealers. We now descrive protocol Π^C_{MPC}, which comprises of four sub-protocols, two for receiving inputs and providing outputs to clients, and two for computing addition and multiplication gates.

10.1 Client Input Protocol

Clients must provide the necessary state encoding their input onto the layer tasked with the evaluation of the first level of the circuit. The protocol is simple because all the difficult guarantees (commitment, linearity among values input by different dealers) are derived from $\mathcal{F}_{\mathsf{VSS}}$, so that the only real task is to produce the polynomial sharing securely even when the client is dishonest. To achieve this, we simply require the client to VSS the coefficients of a polynomial which shares their input, and then we reconstruct each share towards the intended recipient exploiting the VSS linearity. This ensures that all shares lie on a polynomial of degree t even when the dealer is dishonest. We only describe the protocol for one client and one input, but a parallel version where every party in a layer has up to ℓ inputs can be obtained exactly as in Π_{VSS}.

Client Input Protocol Π^k_{Input}

Public Parameters. Clients $\mathbf{C}_1, \ldots, \mathbf{C}_n \in \mathcal{L}_0$. Layer k receiving the input state. The domain M of the inputs from $\mathbf{C}_i$.

Secret Inputs. Each $\mathbf{C}_j$ has input s_j.

Resources. Functionality $\mathcal{F}^{0,\ldots,2k}_{\mathsf{VSS}}$ for $2k$.

Layer $\mathcal{L}_0$: Each committer $\mathbf{C}_j \in \mathcal{L}_0$ does:
- Sample coefficients $\left(f_{1,j}, \ldots, f_{t,j}\right) \leftarrow_{\$} \mathbb{F}^t$.
- Input $(s_j, f_{1,j}, \ldots, f_{t,j})$ to $\mathcal{F}^{0,\ldots,2k}_{\mathsf{VSS}}$.

Layer $\mathcal{L}_{k'}$: Each $P^{k_1}_s \in \mathcal{L}_{k_1}$ does:
- Input $\left((\pi_j, (m^0, \ldots, m^t)), r = P^k_m\right)$ to $\mathcal{F}^{0,\ldots,2k}_{\mathsf{VSS}}$[a] for all $m, j \in [n]$.

Layer $\mathcal{L}_k$: Each $P^k_m \in \mathcal{L}_k$ receives values $(s_{1,m}, \ldots, s_{n,m})$ from $\mathcal{F}^{0,\ldots,2k}_{\mathsf{VSS}}$.

[a] π_i is the linear function denoting the projection on the i-th component. This command reveals $\lambda(f_i(m))$ to P^k_m.

10.2 Secure Multiplication Protocol

Layer $\mathcal{L}_0$ holds the states for the inputs a and b of the multiplication gate g. Each party P^0_i locally multiplies their polynomial shares of a and b and computes $c_i = a_i \cdot b_i$. Then this party reproduces the input state but for the value c_i towards the layer (say $\mathcal{L}_k$) who is tasked to compute the following layer of the circuit. It does so by using the client input protocol Π_{Input}. Notice that $c = a \cdot b$ can be expressed as a linear combination of the c_i's for $i \in [n]$, because $(i, c_i)_{i\in[n]}$ are points on a polynomial $g(x)$ of degree $2t$ (the product of $f_a(x)$ and $f_b(x)$) such that $g(0) = c$, and both polynomial interpolation and polynomial evaluation at 0 are linear functions. Therefore, the state for c can be obtained from the states for each c_i locally.

However, a corrupt $P^0_i \in \mathcal{I}_0$ who inputs a value $\widehat{c}_i \neq a_i \cdot b_i$ can easily disrupt the correctness of this procedure. We therefore require that each party proves, via a distributed ZK-proof, that the input $\widehat{c}_i$ they provided is indeed $\widehat{c}_i = a_i \cdot b_i$. To this end, each party P_i^0 produces new $\mathcal{F}_{\mathsf{VSS}}$ commitments to $\widehat{a}_i, \widehat{b}_i$ and $\widehat{c}_i$, and proves (to parties in some auxiliary layer $\mathcal{L}_{k'}$) that 1) $\widehat{a}_i = a_i$, 2) $\widehat{b}_i = b_i$, and 3) $\widehat{a}_i \cdot \widehat{b}_i = \widehat{c}_i$. If they fail to do so (honest parties never fail in these proofs) then parties in $\mathcal{L}_{k'}$ simply reveal values a_i and b_i to layer $\mathcal{L}_k$. Finally parties in $\mathcal{L}_k$ hold commitments to each c_i (the ones for which the proof fails are just taken as standard states of the reconstructed values) and with this to the product $c = a \cdot b$.

To finish, we just need to explain how parties perform the three required ZK-proofs. Let us first discuss the proof of equality for the commitments to values $\widehat{a}_i$ and a_i performed by P_i^0. To begin the proof of equality, party P_i^0 produces new commitments via $\mathcal{F}_{\mathsf{VSS}}$ to $\widehat{r}_i$ (his claimed version of r_i), towards all future layers until $\mathcal{L}_k$. Then, parties in some later layer (who also hold commitments to a_i, r_i and $\widehat{a}_i$) receive a public random value ρ from functionality $\mathcal{F}_{\mathsf{Beacon}}$. The values $r_i + \rho a_i$ and $\widehat{r}_i + \rho \widehat{a}_i$ are opened publicly, and if they are different the proof fails. Intuitively, if $a_i \neq \widehat{a}_i$ or $r_i \neq \widehat{r}_i$, there is only one value ρ that satisfies the equality, and this can only be guessed with negligible probability if the space of random values sampled by the beacon is large enough.

The proof of correct multiplication to show $\widehat{a}_i \cdot \widehat{b}_i = \widehat{c}_i$ works as follows: party P_i^0 samples a random value β_i and commits (via $\mathcal{F}_{\mathsf{VSS}}$) to values β_i and $b_i\beta_i$. Now, parties in some later layer (who also hold commitments to $\widehat{a}_i, \widehat{b}_i$ and $\widehat{c}_i$) receive a random value ρ from functionality $\mathcal{F}_{\mathsf{Beacon}}$, and publicly open the value $\rho' = \rho a_i + \beta_i$. Finally, a later layer publicly opens the value $\rho' b_i - b_i\beta_i - \rho c_i$ and the proof succeeds if and only if this value is 0. Again, the intuition is that if $\widehat{a}_i \cdot \widehat{b}_i \neq \widehat{c}_i$, there is only one value ρ that satisfies the equality, and the proof succeeds with negligible probability if the value ρ comes from a large enough space. The fresh randomness to be used in the next layer of the circuit is generated in parallel to the multiplication protocol.

Secure Multiplication Protocol $\Pi_{\mathsf{Mult}}^{k;a,b,g}$

Public Parameters A layer $\mathcal{L}_0$ holding the invariant state for input values a and b to a multiplication gate g. A layer $\mathcal{L}_k$ holding the invariant state for output value $c = a \cdot b$ of g. Layers until $\mathcal{L}_{2k}$ hold commitments to c and each polynomial share c_i.

Secret Inputs. Each P_i^0 holds:

- Values $a_i = f_a(i)$ and $b_i = f_b(i)$, where $f_a(0) = a$ and $\deg(f_a) \leq t$ and $f_b(0) = b$ and $\deg(f_b) \leq t$.
- Values $r_i = f_r(i)$ and $r'_i = f_{r'}(i)$ where $f_r(0) = r$ and $\deg(f_r) \leq t$ and $f_{r'}(0) = r'$ and $\deg(f_{r'}) \leq t$.

Resources. Functionality $\mathcal{F}_{\mathsf{Beacon}}$. Protocol Π_{Input}. Functionality $\mathcal{F}_{\mathsf{VSS}}$. Functionality $\mathcal{F}_{\mathsf{VSS}}^{0,\ldots,k}[a,b,r,r']$ such that a,b,r,r' are linear combinations of values input to $\mathcal{F}_{\mathsf{VSS}}^{0,\ldots,k}[a,b,r,r']$ in a previous layers.

Layer $\mathcal{L}_0$: Each P_i^0 does:

- Sample $\beta_i \leftarrow_\$ \mathbb{F}$.
- Compute $c_i = a_i \cdot b_i$.
- Input $(a_i, b_i, r_i, r_i', c_i, \beta_i, b_i \cdot \beta_i)$ to $\Pi_{\mathsf{Input}}^{0,\ldots,2k}$. Denote by $\mathcal{F}_{\mathsf{VSS}}$ the corresponding instance of the functionality.
- Sample $\gamma_i \leftarrow_\$ \mathbb{F}$ and input Π_{Input}^{k}.
- Input sample to $\mathcal{F}_{\mathsf{Beacon}}^{k_1}[j, a]$ for all $j \in [n]$.
- Input sample to $\mathcal{F}_{\mathsf{Beacon}}^{k_1}[j, b]$ for all $j \in [n]$.
- Input sample to $\mathcal{F}_{\mathsf{Beacon}}^{k_1}[j, c]$ for all $j \in [n]$.

Layer $\mathcal{L}_{k_1}$: Each $P_s^{k_1}$ does, for all $i \in [n]$:

- Receive value $\rho_{i,a}$ from $\mathcal{F}_{\mathsf{Beacon}}^{k_1}[i, a]$.
- Receive value $\rho_{i,b}$ from $\mathcal{F}_{\mathsf{Beacon}}^{k_1}[i, b]$.
- Receive value $\rho_{i,c}$ from $\mathcal{F}_{\mathsf{Beacon}}^{k_1}[i, c]$.
- Query $\mathcal{F}_{\mathsf{VSS}}$ to reveal $\rho_{i,a}\widehat{a}_i + \widehat{r}_i$ to $\mathcal{L}_{k_2}$.
- Query $\mathcal{F}_{\mathsf{VSS}}$ to reveal $\rho_{i,b}\widehat{b}_i + \widehat{r}_i'$ to $\mathcal{L}_{k_2}$.
- Query $\mathcal{F}_{\mathsf{VSS}}[a, b, r, r']$ to reveal $\rho_{i,a}a_i + r_i$ to $\mathcal{L}_{k_2}$.
- Query $\mathcal{F}_{\mathsf{VSS}}[a, b, r, r']$ to reveal $\rho_{i,b}b_i + r_i'$ to $\mathcal{L}_{k_2}$.
- Query $\mathcal{F}_{\mathsf{VSS}}$ to reveal $\rho_i' = \widehat{a}_i\rho_{i,c} + \beta_i$ to $\mathcal{L}_{k_2}$.

Layer $\mathcal{L}_{k_2}$: Each party $P_s^{k_2}$ does, for all $i \in [n]$:

- If $\widehat{a}_i\rho_{i,a} + \widehat{r}_i \neq a_i\rho_{i,a} + r_i$ add index i to set $\mathsf{FailProof}$.
- If $\widehat{b}_i\rho_{i,b} + \widehat{r}_i' \neq b_i\rho_{i,b} + r_i'$ add index i to set $\mathsf{FailProof}$.
- Query $\mathcal{F}_{\mathsf{VSS}}$ to reveal $\rho_i'\widehat{b}_i - \widehat{b_i\beta_i} - \rho_{i,c}\widehat{c}_i$ to $\mathcal{L}_{k_3}$.

Layer $\mathcal{L}_{k_3}$: Each party $P_s^{k_2}$ does:

- If $\rho_i'\widehat{b}_i - \widehat{b_i\beta_i} - \rho_{i,c}\widehat{c}_i \neq 0$ then add index i to set $\mathsf{FailProof}$ for all $i \in [n]$
- For all $i \in \mathsf{FailProof}$ query functionality $\mathcal{F}_{\mathsf{VSS}}[a, b, r, r']$ to reveal values a_i and b_i publicly to layer $\mathcal{L}_k$.

Layer $\mathcal{L}_k$: Each P_s^k does:

- Consider standard (constant) sharings of each c_i for $i \in \mathsf{FailProof}$. Let $c_{i,s}$ denote P_s^k share.
- Receive from $\mathcal{F}_{\mathsf{VSS}}$ share $c_{i,s}$ of c_i for each $i \notin \mathsf{FailProof}$.
- Compute c_i as a linear combination $\lambda(c_{1,s}, \dots, c_{n,s})$, where λ is the linear function obtained by composing polynomial interpolation and evaluation at 0.
- Receive from $\mathcal{F}_{\mathsf{VSS}}$ share $\gamma_{i,s}$ of γ_i for each $i \in [n]$.
- Compute $\gamma_s = \sum_{i=1}^n \gamma_{i,s}$.

Lemma 14. *In an execution of* $\Pi_{\mathsf{Mult}}^{k;a,b,g}$ *it holds that* $\Pr[i \notin \mathsf{FailProof} \mid \widehat{c}_i \neq a_i \cdot b_i] = \mathsf{negl}(\kappa)$ *for all* $i \in [n]$.

Proof. We have

$$\begin{aligned}
&\Pr[i \notin \mathsf{FailProof} \mid \widehat{c}_i \neq a_i \cdot b_i] = \\
&\Pr\left[\begin{array}{ll|ll} \rho_i'\widehat{b}_i - \widehat{b_i\beta_i} - \rho_{i,c}\widehat{c}_i = 0 & \vee & \widehat{c}_i \neq \widehat{a}_i \cdot \widehat{b}_i & \vee \\ \widehat{b}_i\rho_{i,b} + \widehat{r}_i' = b_i\rho_{i,b} + r_i' & \vee & \widehat{a}_i \neq a_i & \vee \\ \widehat{a}_i\rho_{i,a} + \widehat{r}_i = a_i\rho_{i,a} + r_i & & \widehat{b}_i \neq b_i & \end{array}\right] \leq \\
&\Pr\left[\rho_i'\widehat{b}_i - \widehat{b_i\beta_i} - \rho_{i,c}\widehat{c}_i = 0 \;\middle|\; \widehat{c}_i \neq a_i \cdot b_i\right] \\
&+ \Pr\left[\widehat{a}_i\rho_{i,a} + \widehat{r}_i = a_i\rho_{i,a} + r_i \mid \widehat{a}_i \neq a_i\right] \\
&+ \Pr\left[\widehat{b}_i\rho_{i,b} + \widehat{r}_i' = b_i\rho_{i,b} + r_i' \mid \widehat{b}_i \neq b_i\right] = \\
&\Pr\left[\rho_{i,c} = \frac{\widehat{b}_i\widehat{\beta}_i - \widehat{b_i\beta_i}}{\widehat{a}_i\widehat{b}_i - \widehat{c}_i}\right] + \Pr\left[\rho_{i,a} = \frac{r_i - \widehat{r}_i}{\widehat{a}_i - a_i}\right] + \Pr\left[\rho_{i,b} = \frac{r_i' - \widehat{r}_i'}{\widehat{b}_i - b_i}\right] = \frac{3}{|\mathbb{F}|},
\end{aligned} \tag{41}$$

where the first inequality follows from a union bound and the last equality from the observation that the three public random values $\rho_{i,c}$, $\rho_{i,b}$ and $\rho_{i,a}$ are independent from

$$(a_i, b_i, r_i, r_i', c_i, \beta_i, b_i \cdot \beta_i). \tag{42}$$

□

10.3 Secure Addition Protocol

Addition gates (and linear gates in general) could be evaluated locally by exploiting the linearity of $\mathcal{F}_{\mathsf{VSS}}$ and of polynomial sharings. However, since the linearity only holds for *parallel* executions of Π_{VSS}, to allow the next layer of parties to compute the next layer of the circuit, we need to "refresh" the state, so that the commitments to the inputs for the next layer are all generated in parallel executions of Π_{VSS}. This can be trivially achieved by multiplying by 1 after performing the addition locally. The state encoding the input of this dummy gate fixed to 1 can be computed as a default state of protocol Π_{Input}. We denote this procedure by Π_{Add}. Notice that this protocol preserves the invariant necessary for the circuit computation.

10.4 Client Output Protocol

The output protocol Π_{Output} is trivial: parties in the layer holding the state corresponding to output c of an output gate g can query $\mathcal{F}_{\mathsf{VSS}}[c]$ to reveal c to the intended recipient (or recipients).

10.5 Circuit Evaluation Functionality and Protocol

The circuit evaluation functionality is straightforward: it receives inputs from parties in Layer $\mathcal{L}_0$ and hands the output of the computation to the intended parties in the last $\mathcal{L}_k$. We refer to the parties in these two layers as the *clients*.

SFE Functionality $\mathcal{F}_f^k$

Public Parameters. Input clients $\mathcal{C}_1, \ldots, \mathcal{C}_n$ in $\mathcal{L}_0$ and output clients $\mathcal{C}'_1, \ldots, \mathcal{C}'_n$ in layer $\mathcal{L}_k$. A function $f : \mathbb{F}^\ell \to \mathbb{F}^{\ell'}$. For each input client $\mathcal{C}_i$ a subset of input indices inputs_i. For each output client $\mathcal{C}'_i$ subset of output indices $\mathsf{outputs}_i$

Secret Inputs. From each client $\mathcal{C}_i$ inputs $(s_i)_{i \in \mathsf{inputs}_i}$ to f.

- Receive inputs $(s_i)_{i \in \mathsf{inputs}_i}$ from all input clients $\mathcal{C}_i$ for $i \in [n]$
- Deliver $f(s_1, \ldots, s_\ell)_{j \in \mathsf{outputs}_i}$ to all output clients $\mathcal{C}'_i$ for all $i \in [n]$.

Below, we present a layered protocol implementing functionality $\mathcal{F}_f^k$ in the hybrid model with $\mathcal{F}_{\mathsf{VSS}}$ and $\mathcal{F}_{\mathsf{Beacon}}$. In addition, and for ease of notation, the protocol invokes the multiplication and inputs protocols described in the previous sections as subroutines.

Circuit Evaluation Protocol Π_{MPC}^{C}

Public Parameters. Input Clients $\mathcal{C}_1, \ldots, \mathcal{C}_n$ in $\mathcal{L}_0$. Output Clients $\mathcal{C}'_1, \ldots, \mathcal{C}'_n$ in $\mathcal{L}_k$. A layered arithmetic circuit with depth d and fan-in 2 over $\mathbb{F}$ computing function $f : \mathbb{F}^\ell \to \mathbb{F}^{\ell'}$. For each input client $\mathcal{C}_i$ a subset of input indices inputs_i. For each output client $\mathcal{C}'_i$ subset of output indices $\mathsf{outputs}_i$

Secret Inputs. From each input client $\mathcal{C}_i$ inputs $(s_i)_{i \in \mathsf{inputs}_i}$ to f.

Resources. Functionality $\mathcal{F}_{\mathsf{VSS}}$. Functionality $\mathcal{F}_{\mathsf{Beacon}}$. Protocol Π_{Input}. Protocol Π_{Mult}. Protocol Π_{Add}. Protocol Π_{Output}.

Layer 0**:** Each input client $\mathcal{C}_i$ inputs $(s_i)_{\mathsf{inputs}_i}$ to Π^k_{Input}.
Layer $\ell \cdot k$**:** In each layer $\mathcal{L}_{\ell \cdot k}$ for $\ell \in [d]$ we have the same invariant:

Invariant:

Each $P_i^{\ell \cdot k}$ holds, for every gate g in layer ℓ of the circuit C with input wires a and b:

- Values $a_i = f_a(i)$ and $b_i = f_b(i)$, where $f_a(0) = a$ and $\deg(f_a) \leq t$ and $f_b(0) = b$ and $\deg(f_b) \leq t$.
- Values $r_i = f_r(i)$ and $r'_i = f_{r'}(i)$ where $f_r(0) = r$ and $\deg(f_r) \leq t$ and $f_{r'}(0) = r'$ and $\deg(f_{r'}) \leq t$[a].

Furthermore $\mathcal{L}_{\ell \cdot d}$ is an auxiliary layer in an instance of functionality $\mathcal{F}_{\mathsf{VSS}}^{(\ell-1)k,\ldots,(\ell+1)k}[a, b, r, r']$ holding commitments to a, b, r, r'.

Each party $P_i^{\ell \cdot k}$ participates in $\Pi_{\mathsf{Mult}}^{(\ell+1)k;a,b,g}$ for each multiplication gate g in the ℓ-th layer of C and in $\Pi_{\mathsf{Add}}^{(\ell+1)k;a,b,g}$ for each multiplication gate g in the ℓ-th layer of C.

Layer $\mathcal{L}_{(d+1)\cdot k}$**:** For every output wire c each party $P_i^{(d+1)\cdot k}$ participates in $\Pi_{\mathsf{Output}}^{(d+1)k+1;c}$ towards each output client $\mathcal{C}_j$ such that $c \in \mathsf{outputs}_j$.

[a]In the first computation layer $\mathcal{L}_k$, we can assume without loss of generality that these random values are computed as sums of additional inputs to the circuit C. They could also generated by means of a layered protocol not involving the clients if asking clients to input randomness is a problem.

10.6 Circuit Evaluation Security

Theorem 1. *If $t < n/2$ and C is a circuit with depth d computing f, the $(n, t, O(d))$ layered protocol Π^C_{MPC} realizes functionality $\mathcal{F}_f$ with $(\mathsf{negl}(\kappa), t)$-statistical security in the $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{Beacon}})$-hybrid model.*

Proof. For any adversary $\mathcal{A}$ we describe a simulator $\sigma(\mathcal{A})$ such that the joint distribution of the view of $\mathcal{A}$ and the output of honest parties in the real world ($\mathcal{A}$ interacting with protocol Π^C_{MPC} and functionalities $\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{Beacon}}$) is statistically close to that the ideal world ($\sigma(\mathcal{A})$ interacting with $\mathcal{F}_f$).

The simulator $\sigma(\mathcal{A})$ behaves as follows: it fixes all inputs of honest clients $\mathcal{C}_i \notin \mathcal{I}_0$ to 0. Then, the simulator executes the protocol on behalf of honest parties and simulating all instances of $\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{Beacon}}$ in accordance with these input states and the messages received from $\mathcal{A}$. Upon receiving the output of corrupted output clients $\mathcal{C}' \in \mathcal{I}_k$, the simulator (that kept track of the view of $\mathcal{A}$) patches the state of honest parties to match this view. Notice that this involves 1) patching polynomial sharings, which is trivial to do, and 2) patching the state of $\mathcal{F}_{\mathsf{VSS}}$, which can be done as explained in Lemma 13.

The adversary, for each layer of the circuit, only learns up to t polynomial shares on polynomials of degree at most t for each value a of each wire in the circuit, and because these sharings are all generated from independent randomness, the view of the adversary in the real and the ideal world is the same (in addition to this, in protocol Π_{Mult} the adversary only sees uniform random values that are independent from honest clients inputs and can be perfectly simulated by $\sigma(\mathcal{A})$).

To argue that the outputs of honest clients are statistically close in the real and ideal world, notice that 1) in the ideal world the outputs of honest clients are simply the outputs of $\mathcal{F}_f$ computed on the real inputs of honest clients, while 2) in the real world the output of honest clients are computed according to the outputs of $\mathcal{F}_{\mathsf{VSS}}$ for the output wires of C. Thanks to the linearity of $\mathcal{F}_{\mathsf{VSS}}$ and of polynomial shares, these values are exactly the outputs of f *unless* $\mathcal{A}$ manages to break one of the multiplication proofs for *some* gate g in any layer ℓ of circuit C for some corrupted parties in $P_i^{\ell \cdot k}$. Thanks to Lemma 14 this only happens with probability $|C| \cdot t \cdot \mathsf{negl}(\kappa)$, which is again negligible in the security parameter. $\square$

References

[1] Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. SCALES - MPC with small clients and larger ephemeral servers. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 502–531. Springer, Heidelberg, November 2022.

[2] Zuzana Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 305–328. Springer, Heidelberg, March 2006.

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

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

[5] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 353–380. Springer, Heidelberg, November 2020.

[6] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.

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

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

[9] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[10] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

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

[12] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 311–326. Springer, Heidelberg, May 1999.

[13] Bernardo David, Giovanni Deligios, Aarushi Goel, Yuval Ishai, Anders Konring, Eyal Kushilevitz, Chen-Da Liu-Zhang, and Varun Narayanan. Perfect MPC over layered graphs. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 360–392. Springer, Heidelberg, August 2023.

[14] Giovanni Deligios, Aarushi Goel, and Chen-Da Liu-Zhang. Maximally-fluid mpc with guaranteed output delivery. Cryptology ePrint Archive, Paper 2023/415, 2023. `https://eprint.iacr.org/2023/415`.

[15] Yvo Desmedt and Yongge Wang. Perfectly secure message transmission revisited. In Lars R. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, pages 502–517, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[16] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *Journal of the ACM (JACM)*, 40(1):17–47, 1993.

[17] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *33rd ACM STOC*, pages 580–589. ACM Press, July 2001.

[18] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In Brian A.

Coan and Yehuda Afek, editors, *17th ACM PODC*, pages 101–111. ACM, June / July 1998.

[19] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Heidelberg.

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

[21] Vipul Goyal, Elisaweta Masserova, Bryan Parno, and Yifan Song. Blockchains enable non-interactive MPC. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 162–193. Springer, Heidelberg, November 2021.

[22] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, January 2000.

[23] Martin Hirt, Ueli M. Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 143–161. Springer, Heidelberg, December 2000.

[24] Dennis Hofheinz and Jörn Müller-Quade. A synchronous model for multi-party computation and the incompleteness of oblivious transfer. In *Proceedings of FCS*, pages 117–130. Citeseer, 2004.

[25] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.

[26] Leslie Lamport, Robert Shostak, and Marshall Pease. *The Byzantine generals problem*, page 203–226. Association for Computing Machinery, New York, NY, USA, 2019.

[27] Chen-Da Liu-Zhang, Elisaweta Masserova, João Ribeiro, Pratik Soni, and Sri AravindaKrishnan Thyagarajan. Improved yoso randomness generation with worst-case corruptions. In *Financial Cryptography and Data Security – FC 2024*, 2024.

[28] Chen-Da Liu-Zhang and Ueli Maurer. Synchronous constructive cryptography. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 439–472. Springer, Heidelberg, November 2020.

[29] Pasin Manurangsi, Akshayaram Srinivasan, and Prashant Nalini Vasudevan. Nearly optimal robust secret sharing against rushing adversaries. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 156–185. Springer, Heidelberg, August 2020.

[30] Jesper Buus Nielsen. *On protocol security in the cryptographic model.* BRICS, 2003.

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

[32] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In Luigi Logrippo, editor, *10th ACM PODC*, pages 51–59. ACM, August 1991.

[33] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

[34] Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 719–749. Springer, Heidelberg, August 2022.

[35] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

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