Perfect MPC over Layered Graphs

Bernardo David[1], Yuval Ishai[2], Anders Konring[1], Eyal Kushilevitz[2], and Varun Narayanan[2]

[1] IT University of Copenhagen, Denmark
[2] Technion - Israel Institute of Technology, Israel

Abstract. The classical "BGW protocol" (Ben-Or, Goldwasser and Wigderson, STOC 1988) shows that secure multiparty computation (MPC) among n parties can be realized with *perfect full security* if $t < n/3$ parties are corrupted. This holds against malicious adversaries in the "standard" model for MPC, where a fixed set of n parties is involved in the full execution of the protocol. However, the picture is less clear in the mobile adversary setting of Ostrovsky and Yung (PODC 1991), where the adversary may periodically "move" by uncorrupting parties and corrupting a new set of t parties. In this setting, it is unclear if full security can be achieved against an adversary that is maximally mobile, *i.e.,* moves after every round. The question is further motivated by the "You Only Speak Once" (YOSO) setting of Gentry *et al.* (Crypto 2021), where not only the adversary is mobile but also each round is executed by a disjoint set of parties. Previous positive results in this model do not achieve perfect security, and either assume probabilistic corruption and a nonstandard communication model, or only realize the weaker goal of security-with-abort. The question of matching the BGW result in these settings remained open.

In this work, we tackle the above two challenges simultaneously. We consider a *layered MPC* model, a simplified variant of the fluid MPC model of Choudhuri *et al.* (Crypto 2021). Layered MPC is an instance of standard MPC where the interaction pattern is defined by a layered graph of width n, allowing each party to send secret messages and broadcast messages only to parties in the next layer. We require perfect security against a malicious adversary who may corrupt at most t parties in each layer. Our main result is a perfect, fully secure layered MPC protocol with an optimal corruption threshold of $t < n/3$, thus extending the BGW feasibility result to the layered setting. This implies perfectly secure MPC protocols against a maximally mobile adversary.

1 Introduction

The goal of classic Secure Multiparty Computation (MPC) protocols is for a set of n mutually distrusting parties to jointly compute a function on their secret inputs without revealing anything but the output of the function. The protocols are typically run in the presence of an adversary and security is guaranteed if no more than t out of the n parties in the system are compromised for the duration of the *entire protocol.* In this setting, the well known result by Ben-or, Goldwasser and Wigderson [BGW88] (BGW) shows that it is possible to achieve *perfect full security* when $t < n/3$, *i.e.* security against an unbounded active adaptive adversary corrupting $t < n/3$ parties with guaranteed output delivery (G.O.D.).

Inspired by real-world scenarios with long-running computations where parties may recover from corruptions, Ostrovsky and Yung [OY91] put forward a notion of a mobile adversary that is able to compromise all parties *eventually* but is limited to a threshold of t out of n parties at any given time. In this setting, an execution is divided in rounds that are grouped into epochs. The adversary can "move" at the onset of every epoch by choosing a new set of parties to corrupt and remains static for the remainder of the epoch. Former corrupted parties are "rebooted" into a clean initial state (or, equivalently, update their internal state and securely erase past state). In [OY91], it is proven that there exists a fully secure proactive MPC protocol in the presence an active mobile adversary but allowing only a small constant fraction of corrupted parties. Subsequent works [HJKY95, ADN06, BELO15, ELL20] explored more efficient protocols with other security guarantees under further restrictions to the mobile adversary but still fell short of 1-round epochs or achieving the optimal corruption threshold $t < n/3$ of BGW.

Departing from the player replaceability[3] and anonymous committees of distributed ledgers, the notion of You Only Speak Once (YOSO) MPC (introduced in [GHK+21]) takes proactive security one step further, by having a freshly elected anonymous committee of parties execute each round of the protocol. As an extra restriction, parties are only allowed to send messages once (*i.e.* when they

[3] A term from [GHM+17] for protocols where a new set of parties executes each round.

execute their role in the protocol). However, YOSO assumes parties can use ideal *target-anonymous channels* to send messages to parties who are elected to execute a role in any future round without learning their identities. The fact that each round is executed by anonymous parties elected at random turns the corruption model probabilistic: even though an adaptive adversary may corrupt any party at any time (up to a corruption threshold t), it only successfully corrupts a party executing a certain round with some small constant probability (given that committees are large enough). In this setting, it was shown [GHK$^+$21] that statistically secure MPC with G.O.D. is possible when the adversary corrupts $t < n/2$ parties, albeit not for constant n due to probabilistic corruptions. Fluid MPC [CGG$^+$21] is a variant of this model without target-anonymous channels where parties may act in more than one round before being substituted, but their results fall short of full security, as they do not achieve G.O.D.. Another variation was shown in SCALES [AHKP22], which allows for special clients who provide an input and receive an output to act in more than one round (while server committees may only act once), focusing on protocols with computational security.

Inspired by the original mobile adversary characterized by [OY91] and the recent line of work on MPC with dynamic committees [GHK$^+$21, CGG$^+$21, AHKP22], we ask again the question originally settled in BGW [BGW88] but now in a more challenging setting:

Is it possible to construct MPC with dynamic committees achieving perfect full security against an adaptive rushing adversary and with optimal corruption threshold?

1.1 Our Contributions

Layered MPC. We first define layered MPC, which captures the most stringent setting in the intersection of the mobile adversary and the YOSO models. In layered MPC, parties communicate through a directed layered graph of d layers corresponding to each protocol round. Each round is executed by a unique set of n parties sitting at a layer, which is disjoint from all other sets of parties in other layers. Parties in one layer can only receive messages from parties in the immediately previous layer and send messages to the parties in the immediately next layer. We consider an active, adaptive, rushing adversary that corrupts up to t out n parties in each layer. We write (n, t, d)-layered MPC as shorthand for a layered MPC protocol with d layers (*i.e.* rounds) of n parties out of which t may be corrupted. We provide a formalization of this model and show that layered MPC protocols can be analyzed within well established frameworks such as the real/ideal world paradigm [Can00, Gol09] and Universal Composability [Can01].

Layered MPC is similar to fully Fluid MPC [CGG$^+$21] with parties only executing one round. We show that a secure layered MPC protocol is also secure against a maximally mobile adversary [OY91] that moves after every round. In comparison to YOSO [GHK$^+$21], layered MPC imposes stronger restrictions on honest parties, who cannot receive a message from a party in an arbitrary past committee or send a message to a party in an arbitrary future committee. Moreover, the adversary is not restricted to probabilistic corruptions but is limited to corrupting t out n parties in each layer, allowing for threshold-optimal protocols.

Main Results. In Section 3 we construct basic primitives that help realize layered VSS based on CNF[4] (replicated) secret sharing. We present a nontrivial adaptation of a VSS protocol of Gennaro et al. [GIKR01] to the layered setting. The main challenge is to eliminate the repeated interaction between the parties and the dealer, which is not possible in the layered setting. While CNF-based protocols scale exponentially with n, they are simpler than the Shamir-based counterparts that we will present next, and can have efficiency advantages for small values of n, especially when settling for computational security.

Theorem 1 (CNF-Based Layered VSS). *For any n, t such that $t < n/3$, and $d \geq 5$, there exists an (n, t, d)-layered MPC protocol realizing CNF-VSS. For $d = O(1)$ and secrets of length ℓ, the protocol requires $\ell \cdot 2^{O(n)}$ bits of communication, counting both point-to-point messages and broadcast. When settling for* computational *security with perfect correctness and using a black-box PRG with seed length λ, there is a protocol with $\lambda \cdot 2^{O(n)} + O(n\ell)$ bits of communication.*

[4] In CNF-based secret sharing, the secret is first split into $\binom{n}{t}$ additive shares–a share r_T for each set $T \subset [n]$ of size t–and party i receives all shares r_T such that $i \notin T$.

In Section 4 we build on the above VSS protocol to obtain a *general* layered MPC protocol based on CNF secret sharing. The protocol applies to layered arithmetic circuits, in which each layer of the circuit only takes inputs from the previous layer. Every circuit of depth D can be converted to a layered circuit with D layers, incurring at most a quadratic but typically (nearly) linear overhead to the circuit size. Building on a constant-round protocol from [DI05], in Section 6 we describe how to amortize the overhead of CNF secret sharing by settling for computational security.

Theorem 2 (CNF-Based Layered MPC). *Let f be an n-party functionality computed by a layered arithmetic circuit C over a finite ring, with D layers and M gates. Then, for any $t < n/3$, there is an $(n, t, O(D))$-layered MPC protocol for f. The communication consists of $2^{O(n)} \cdot M$ ring elements. Alternatively, settling for* computational *security with perfect correctness and using a black-box PRG with seed length λ, there is a $(n, t, O(1))$-layered MPC protocol for a Boolean circuit (i.e., the ring is $\mathbb{F}_2$) with M gates with $\lambda \cdot 2^{O(n)} + O(n^5 \cdot M)$ bits of communication.*

While the CNF-based protocols are relatively simple and have concrete efficiency benefits for small values of n, they do not yield a general feasibility result that scales polynomially with n. In Section 5 we establish such a result using (the bivariate version of) Shamir's secret-sharing scheme.

Theorem 3 (Efficient Layered MPC). *Let f be an n-party functionality computed by a layered arithmetic circuit C over a finite field $\mathbb{F}$, with D layers and M gates. Then, for any $t < n/3$, there is a polynomial-time $(n, t, O(D))$-layered MPC protocol for f. More concretely, the communication consists of $M \cdot O(n^9)$ field elements.*

Proactive MPC. The original concept of proactive MPC put forward by [OY91] considered an adversary that has the ability to move in *every* round of the protocol. We define such an adversary in Definition 2 and label it *maximally mobile* while protocols that can thwart such an adversary are called maximally proactive. We show that a secure layered MPC protocol is a maximally proactively secure protocol in Lemma 1. This allows us to extend our security analysis from the layered to the proactive setting. We define maximally Proactive Secret Sharing and MPC in Definition 5 and provide the following threshold-optimal results by combining Theorem 3 and Lemma 1.

Corollary 1 (Perfectly Secure Maximally Proactive MPC). *Let f be an n-party functionality computed by a layered circuit C over a field $\mathbb{F}$, with D layers. Then, for $t < n/3$, there is an efficient maximally proactive MPC protocol computing f in $r = O(D)$ rounds.*

Secure Message Transmission and Broadcast. Sending a message to a party that acts in an arbitrary future round is a recurring problem in settings such as layered MPC. In YOSO [GHK+21] it is circumvented by assuming target-anonymous channels, an ideal resource that allows a party in round r to send a message to a party who is elected to perform a certain role in round $r' > r + 1$ without learning its identity. We take steps to obtain a similar primitive (although without anonymity guarantees) by relying only on the parties in the layered graph to carry the message forward, despite our much more restrictive interaction pattern that precludes such communication. In Section 3.1 we provide a thorough analysis of an important primitive in layered MPC called *Future Messaging*. The functionality f_{FM} is described in Functionality 3.1. Future Messaging takes as input a message m from a sender in $\mathcal{L}_0$ and, if the sender is honest, the message m arrives at the recipient. In the context of layered MPC this primitive is close to an instance of 1-way Secure Message Transmission (SMT) over a directed graph. We show that it is possible to self-compose this primitive to carry a message from a sender in $\mathcal{L}_0$ to a designated receiver in $\mathcal{L}_d$ for $d > 1$. The following theorem characterizes our construction.

Theorem 4 (Restatement of Theorem 6). *For any $d > 0$, any n and t where $t < n/3$, and message domain M, there exists a protocol Π_{FM} that realizes f_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$ with perfect t-security and communication complexity $O(n^{\lceil \log d \rceil} \log |M|)$.*

Using the layered protocol for Shamir VSS and resharing, which we construct building on Future Messaging, we can make the dependence of the communication cost of Future Messaging on d linear. This is achieved by having the sender verifiably secret the message using VSS and then reshare it repeatedly until reaching the layer previous to that of the receiver, at which point the shareholders of the value can reveal the message to the receiver by transferring all its shares. Communication cost of

VSS and of resharing across a constant number of layers is $\mathsf{poly}(n)$, making the communication cost of Future Messaging linear in d.

The layered model allows for layer-to-layer broadcast. That is, any party in $\mathcal{L}_a$ may broadcast to parties in $\mathcal{L}_{a+1}$. It turns out that this assumption is necessary, since we prove that any deterministic broadcast in the setting of layered MPC is possible only if $t = 0$. Our analysis is shown in Appendix A.3, where we cast the result of [Gar94] to the setting of layered MPC and obtain the following result.

Theorem 5. *Deterministic perfect Broadcast in the setting of layered MPC is possible iff* $t = 0$.

1.2 Related Work

We summarize the relationship between previous works in similar settings and our results in Table 1.2. We discuss further related works below.

Results for Maximally Proactive MPC with Dynamic Committees					
Functionality	Reference	Level	Security	Complexity	Threshold
Future Messaging	Section 3.1	perfect	full	$\mathsf{poly}(n)$	$t < n/3$
VSS	[BGG+20]	computational	full	$\mathsf{poly}(n)$	$t < n/4^*$
	Section 4.2	perfect	full	$2^{O(n)}$	$t < n/3$
	Section 5	perfect	full	$\mathsf{poly}(n)$	$t < n/3$
MPC	[GHK+21] (YOSO)	statistical	full (w/setup†)	$\mathsf{poly}(n)$	$t < n/2^*$
	[CGG+21] (Fluid)	statistical	w/abort	$\mathsf{poly}(n)$	$t < n/2$
	[OY91]	perfect	full	$\mathsf{poly}(n)$	$t < n/d$
	Section 4.4	perfect	full	$2^{O(n)}$	$t < n/3$
	Section 5	perfect	full	$\mathsf{poly}(n)$	$t < n/3$

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)

Proactive Secret Sharing (PSS). PSS protocols aim at solving the problem that shares learned by the adversary are compromised forever by resharing the secret periodically. The *static group* setting where resharing is done among the same set of parties is considered in [HJKY95, CH01, ADN06, BELO15]. However, this is often insufficient since it assumes a world where a server never fails to the extend that it cannot recover again. The setting of *dynamic groups* where resharing is done towards a different (possibly disjoint) set of parties is considered in [DJ97, WWW02, ELL20]. Finally, proactive techniques in asynchronous settings have been treated in [CKLS02, SLL10].

Permissionless Networks. In the context of permissionless networks where parties are allowed to join and leave as they wish, the *dynamic group* property has taken on a new meaning. The recent focus on this setting spurred new interest in (dynamic) proactive techniques [MZW+19, GKM+22]. Particularly interesting, is the definition of evolving committee secret sharing [BGG+20] that places the responsibility of keeping a tolerable corruption threshold on the protocol designer.

Maximally PSS and MPC with Dynamic Committees. Recently, a number of works [GHM+17, GHK+21, CGG+21, AHKP22] have considered extreme settings with dynamic committees, where each round of a protocol is executed by a new set of parties considering maximally mobile (or even adaptive) adversaries. In YOSO [GHK+21], an ideal mechanism guarantees that a set of anonymous parties is selected at random to execute each round, effectively limiting the adversary to probabilistic corruptions. Hence, YOSO is incompatible with settings where n and t are constant. Moreover, parties have access to ideal target-anonymous channels allowing for communication to *any* party in the future. Hence, results in the YOSO model do not directly translate to our setting even if we settle for non-optimal corruption thresholds, as YOSO protocols may crucially rely on the ability to send messages across many layers. For example, in the information theoretical signature protocol of [GHK+21, Section 3.3], a cut-and-choose mechanism is realized assuming that a sender can commit to a set of message authentication codes (MACs) by sending them directly to a receiver, after which verifiers broadcast random subsets of keys, which the receiver uses to check these MACs. The security of this technique crucially relies on the fact that using ideal target-anonymous channels guarantees that the sender cannot changes the MACs sent to the user *after* the verifiers announce the checking keys. This technique does not work in the layered MPC setting with our weaker Future Messaging protocol, which does not commit a corrupted sender to the messages it transmits to future layers.

Closest to layered MPC is Fluid MPC [CGG$^+$21] in its most extreme configuration (fully fluid), where parties can execute a single round of the protocol and immediately leave but are not necessarily selected anonymously and at random. Curiously, one of the goals of Fluid MPC is maintaining a small state complexity. In particular, the computation and communication of each committee in Fluid MPC is independent of the size of the circuit. While this is attractive, we do not make any such claims and we also only consider already layered circuits[5]. Finally, a crucial difference is that the protocols presented for Fluid MPC only enjoy security-with-abort while we aim for full security.

While the use of an arbitrary interaction pattern in layered MPC is similar to [HIJ$^+$16], our focus is on a specific interaction pattern capturing extreme cases of MPC with dynamic committees and a maximally mobile adversary.

1.3 Technical Overview

The goal of this paper is to build a layered MPC protocol that takes inputs from a set of clients in the input layer and securely delivers a function of the inputs to a set of output clients in a later layer. For $t < n/3$, we present two layered protocols for general MPC with t-security: a simple but inefficient construction based on CNF secret sharing and a more complex but efficient construction based on Shamir secret sharing.

Owing to a highly restrictive communication pattern and the presence of a very powerful adversary, implementing layered MPC with optimal corruption threshold presents several interesting challenges. The most apparent is the complete prohibition of interaction, as parties executing the protocol do not persist. We emulate a limited kind of interaction by having a party who wants to speak a second time hide all possible messages it may want to convey in a future layer and selectively reveal the appropriate message to the next layer. In such cases, it is imperative to the security of the party that only the appropriate message is revealed while the other messages are effectively destroyed. Interestingly, realizing this limited form of interaction takes us a long way in implementing layered MPC. This leads us to the first primitive we construct in this presentation:

Future Messaging. Future messaging allows a party (sender) to securely send a message to another party (receiver) situated in a later layer. To send a message two layers down, the sender can secret share the message onto the next layer using any t-secure secret sharing scheme; parties in the next layer can then forward these shares to the receiver who can recover the message by robust reconstruction of the received shares. We extend this intuition to allow a sender to securely send a message to a designated receiver in any future layer. This protocol is non-commiting; hence, a corrupt sender can choose the message to deliver to the receiver based on the adversary's view until the layer in which the receiver is situated. Effectively, future messaging allows rushing till the receiver's layer! Future messaging allows a sender to distribute a secret sharing of a value onto a future layer; parties in this layer can disclose this value to a receiver (or broadcast it to all parties) in the next layer based on a unanimous decision (potentially depending on computation that was carried out in an intermediate layer). In this manner, we emulate the aforementioned (limited) interaction by the sender.

MPC using CNF Shares. Equipped with a protocol for future messaging, we set out to build a layered protocol for verifiable secret sharing (VSS). We will then follow the standard approach for secure function evaluation, where a layered arithmetic circuit computing the function is evaluated by progressively and securely computing secret shares of the value on the output wire of each gate using the secret shares of the values on the input wires, finally revealing the values on the output wires of the circuit to the output clients.

Verifiable CNF secret sharing. To achieve verifiable CNF secret sharing, it suffices to implement a seemingly simpler primitive, namely future multicast, which allows a dealer to securely send a message to a designated subset of receivers in a later output layer with the guarantee that all receivers get the same message even if the sender is corrupt. Verifiable CNF secret sharing is achieved by having

[5] The inherent issue with state complexity originates from a common misconception (see fx [DEP21]) that *any* general arithmetic circuit can be transformed into a layered circuit with same depth and only linear overhead in width.

the dealer split the secret into $\binom{n}{t}$ additive shares (a share r_T for each $n-t$ sized set $T \subset [n]$) and multicast r_T to all output clients in T.

While implementing multicast, we encounter many challenges inherent to layered MPC. When realizing multicast, the sender sends the same message to a (sub)set of parties in the next layer, who raise a complaint if they receive distinct messages, in which case the sender publicly discloses the message. Clearly, this sequence of interactions is non-trivial to realize in a layered network, where the sender cannot speak a second time and the parties in a layer cannot communicate with each other. Hence, we use a weak notion of secure addition (See Section 3.2) to allow the receiving parties to securely reveal the difference between the values they received to all parties two layers down. If the difference is non-zero for any pair of values, the layer that learns this difference collectively decides to disclose the sender's message using the trick we previously outlined.

Having implemented verifiable CNF secret sharing, we proceed to secure computation of arithmetic gates. Since the secret sharing is linear, addition and multiplication-by-constant gates can be computed by local processing, which leaves us with the secure computation of the multiplication gate that takes the secret shares of two values and computes a secret sharing of their product.

Multiplication. Our layered protocol for multiplication is built by porting the classic protocol for secure multiplication in the standard (non-layered) setting. In this process, we face all the challenges we encountered while realizing future multicast. Suppose a value is secret shared on a layer and is also required in another layer. Naively replicating the same share in the later layer is insecure since the adversary can reconstruct the secret by corrupting t parties in each of these layers and obtaining $2t$ shares. We get around this problem with a simple trick that avoids using a full-fledged protocol for resharing CNF shares.

We realize secure computation by evaluating a layered arithmetic circuit using the protocols we constructed so far. To properly process the layered circuit, we rely on the invariant that the secret shares of the values on all the input wires to any layer of the circuit are simultaneously available on the same layer of the layered network. However, secret shares of the output of a linear gate (addition or multiplication-by-constant) can be computed locally while those of a multiplication gate using our protocol consume several layers. To keep the invariant, we need the outputs of the linear gates to be available on the output layer of multiplication. Once again, the shares of the outputs cannot be naively secret shared. Instead, we attach a multiplication gate to the output wire of linear gate that takes identity as the other input; this ensures that the shares of the values on all output wires are available simultaneously on the same layer.

Composability of layered protocols. We use simpler layered protocols as subroutines for building more complex ones. For example, the multiplication protocol uses a protocol for verifiable secret sharing (among others) as a subroutine. Hence, it is necessary that the concurrent execution of layered protocols preserve their security guarantees under concurrent composition. We refrain from first proving UC security of our building blocks and then using modular composition theorems since such an analysis will be cumbersome over a synchronous layered graph. Instead, we prove the security of our protocols by constructing simulators and carefully arguing their security. We establish game based properties of layered protocols that are preserved when they are used as subroutines and prove the security using hybrid arguments that exploit these properties. Finally, a few of our constructions make exclusively sequential (non-concurrent) calls to subroutines that have been proven to be standalone secure; in such instances, we use the sequential composition theorem of Canetti [Can00] to argue security (see the security proofs for future messaging and secure function evaluation protocols).

Efficient MPC using Shamir Secret Sharing. We build layered protocols whose communication complexity scales polynomially with the number of parties per layer. This is achieved by porting the cannonical secure function evaluation protocol using Shamir secret shares into the layered model. In order to achieve this, we first develop a layered protocol for verifiable Shamir secret sharing.

Verifiable Shamir secret sharing. We "port" the classic protocol for VSS in the standard setting to the layered setting using the tools we developed in the previous sections along the way to tackle the usual challenges faced in the process. At the end of this process, the parties in the layer right after the input layer hold the purported shares of the dealer's secret and parties 5 layers down publicly hold the updates to the purported shares such that, they together form a valid secret sharing. The parties cannot transfer these shares to the shareholders in the output layer without causing duplication. To get around this, the dealer secret shares coefficients of random a degree t polynomial they wish to use for Shamir secret sharing; the evaluation of the polynomial at distinct points is computed using linear operations and securely delivered to the shareholders in the output layer. This ensures privacy of the secret when the dealer is honest.

Equipped with a layered protocol for Shamir VSS, we use known techniques to realize resharing which allows a layer holding valid shares of a value to securely deliver fresh shares of the same value to a later layer. Using VSS and resharing, porting protocols for secure multiplication and then secure function evaluation into the layered setting is relatively straightforward. We depart form the protocol for general MPC provided in [CDN15]. The protocol uses a form of reinforced secret sharing where the shares of a secret are further secret shared among the shareholders, which is straightforward to implement using VSS and resharing.

2 Preliminaries

2.1 Layered MPC

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; (2) the adversary can corrupt at most t parties in each layer. This is illustrated in Figure 1 and formalized below.

Definition 1 (Layered MPC). *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 $\mathcal{L}_i$, $0 \le i \le d$, 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. By default, 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, adaptive*[6] *adversaries.*

We say that a protocol Π is a layered MPC protocol for f *if it realizes f in the standard sense of (standalone) secure MPC with general adversary structures [Can00, Gol09, HM00]. We require* perfect full security *(with guaranteed output delivery).*

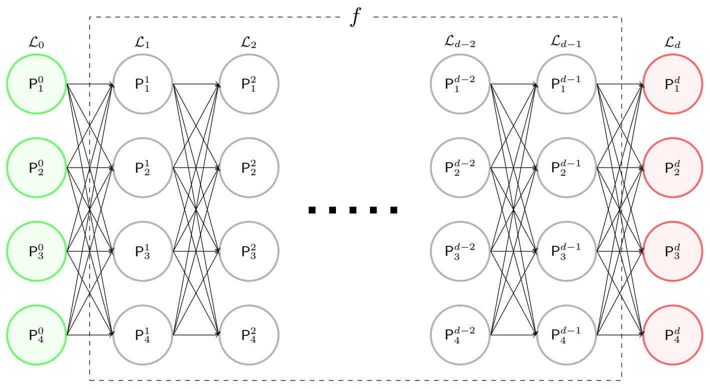


Fig. 1. Layered MPC computing function f with $n = 4$

[6] In the coming sections our security analysis is with respect to non-adaptive adversaries for simplicity. In Section 2.3 we justify this leap appealing to the work of [CDD$^+$04].

Remark 1 (Generalized layered MPC). The above definition is meant to give the simplest formalization of the core problem we study. It can be naturally extended to allow a different number of parties n_i and a different corruption threshold t_i in each layer (our main feasibility result extends to the case where $t_i < n_i/3$), and to allow inputs and outputs from parties in intermediate layers. Our strict notion of perfect full security can also be relaxed in the natural ways. In some cases, we will present efficiency improvements that achieve *computational* (full) security with perfect correctness, meaning that the effect of a computationally unbounded adversary on the outputs of honest parties can be perfectly simulated.

The need for ideal broadcast: In Appendix A.3 we show that broadcast for layered MPC is impossible if $t > 0$. Hence, we must assume ideal broadcast.

Layered MPC implies Proactive Security: In Appendix 2.2 we precisely define maximally proactive security and prove that it is implied by layered MPC.

2.2 Security in Layered MPC Implies Proactive Security

The original definition of a Mobile Adversary [OY91] gives the adversary t pebbles at the outset of the protocol. It can then place the pebbles freely at the beginning of every round among the parties involved where a pebble represents fully corrupting a party. If a pebble is removed from a party, at the beginning of next time period the party will "reboot" into a pre-specified state and its random tape will be renewed. As such, this pebble game represents the race between corruption and recovery and results in a system where a bound is given on the number of corrupted parties which holds in each time period, but in each period the set of corrupted parties can change. We now define the mobile adversary and the execution of a protocol in the context of a mobile adversary.

Definition 2 (Mobile Adversary). *A (t, ρ)-mobile adversary, with corruption threshold t and roaming speed (mobility) ρ, is an adversary that can corrupt at most t parties in each time period where a time period consists of ρ rounds. If $\rho = 1$, we say that the adversary has maximal roaming speed or is a maximally mobile t-adversary.*

Definition 3 (Maximally Proactive Security). *Let r be the round number and T_r the set of at most t parties that is corrupted in round r. In each round the parties can communicate using pairwise secure point-to-point channels or broadcast. An execution of a protocol Π with R rounds of communication in the presence of a maximally mobile t-adversary proceeds as follows.*

0. All parties receive inputs. Adversary chooses an initial set T_0 of t parties to corrupt.
1. Initialize a round counter $r = 1$.
2. Parties send messages of round r (including to themselves), and honest parties update their state.
3. Adversary chooses a new set T_r of t parties to corrupt.
4. Parties receive messages of round r.
5. If $r < R$, increment r and go to (2). Otherwise, honest parties compute an output from messages received in round R.

A protocol that is secure when executed in the presence of a maximally mobile t-adversary is called maximally proactive.

We wish to characterize the relationship between layered and maximally proactive security. Intuitively it is clear, that each layer in the context of layered MPC represents a new round in a maximally proactive protocol and the maximally mobile adversary corrupts a (possibly) new set of parties in every round. In Lemma 1 we capture this intuition in a more formal way. Of course, a necessary assumption is the protocol's ability to include special erasure instructions whereby parties remove sensitive data from the their local state.

Lemma 1 (Layered and Maximally Proactive Security). *Secure layered MPC implies secure maximally proactive MPC under the assumption of secure erasures.*

Proof. Let Π be an (n, t, d)-layered protocol for computing the functionality f with $N = n(d + 1)$ parties partitioned into layers $\mathcal{L}_0, \ldots, \mathcal{L}_d$ (Definition 1). And let $\mathcal{A}_{\mathsf{mobile}}$ be a maximally mobile t-adversary (Definition 2). We prove the implication by constructing a simulator $\mathcal{S}$ that perfectly emulates $\mathcal{A}_{\mathsf{mobile}}$, effectively reducing security for maximally proactive MPC to security for layered MPC.

First, we argue that the interaction pattern induced by layered MPC is equivalent to that of the virtual model (transmission graph) of maximally proactive MPC. Consider the parties $\mathsf{P}_1^0, \ldots, \mathsf{P}_n^d$ computing Π and assume a maximally proactive setting with n parties labeled as $\mathsf{Q}_1, \ldots, \mathsf{Q}_n$. Due to the assumption of secure erasure, we can associate a virtual party Q_i^r with each round $0 \leq r < d$. Unless, Q_i is corrupted in both round $r-1$ and r, the party Q_i^{r-1} shares no state with Q_i^r apart from the messages received over broadcast or secure point-to-point channels. Similarly, from the definition of layered MPC P_i^r shares no state with P_i^{r-1} unless both are corrupted. Finally, we observe that the interaction pattern between parties $\mathsf{P}_1^0, \ldots, \mathsf{P}_n^d$ is equivalent to the interaction pattern induced by the virtual model connecting the virtual parties $\mathsf{Q}_1^0, \ldots, \mathsf{Q}_n^d$.

We now sketch the simulator $\mathcal{S}$. Let $\mathcal{A}_{\mathsf{layered}}$ be the following (n, t, d)-layered MPC adversary during an execution of protocol Π. $\mathcal{A}_{\mathsf{layered}}$ runs $\mathcal{A}_{\mathsf{mobile}}$ internally and sets $r = 0$ before the first round starts. Then, $\mathcal{A}_{\mathsf{layered}}$ does the following, for each $0 \leq r \leq d$: (1) it receives from $\mathcal{A}_{\mathsf{mobile}}$ a set of parties to corrupt $T_r = \{\mathsf{Q}_0^r, \ldots, \mathsf{Q}_t^r\}$ (2) it corrupts corresponding parties $\{\mathsf{P}_0^r, \ldots, \mathsf{P}_t^r\}$ in the execution of Π. (3) it returns the state of the corrupted parties in round r to $\mathcal{A}_{\mathsf{mobile}}$.

Since Π is a protocol for layered MPC there exists a simulator $\mathcal{S}_{\mathsf{layered}}$ for $\mathcal{A}_{\mathsf{layered}}$ which we will use to construct our final simulator $\mathcal{S}$. $\mathcal{S}$ runs $\mathcal{S}_{\mathsf{layered}}$ internally in the following way for each round $0 \leq r \leq d$. (1) $\mathcal{S}$ receives from $\mathcal{S}_{\mathsf{layered}}$ a request to corrupt parties $\{\mathsf{P}_0^r, \ldots, \mathsf{P}_t^r\}$. (2) $\mathcal{S}$ sends to the functionality a request to corrupt $T_r = \{\mathsf{Q}_0^r, \ldots, \mathsf{Q}_t^r\}$. (3) $\mathcal{S}$ provides the state of the parties from the functionality to $\mathcal{S}_{\mathsf{layered}}$. Finally, $\mathcal{S}$ outputs whatever $\mathcal{S}_{\mathsf{layered}}$ outputs.

We note that while Lemma 1 characterizes a strong relation between the layered MPC model and security in the presence of a maximally mobile adversary, the existing literature generally considers proactive security against a slower-moving adversary. In [ADN06, BELO15, MZW$^+$19, ELL20], the protocol time-line is split into phases where each protocol round belongs to exactly one phase and between each pair of consecutive phases a refresh protocol is run to ensure re-randomization and redistribution of the secret. Typically, the adversary can then adaptively corrupt at most t parties between the start of one refresh until *the end* of the next, effectively, halving the mobile adversary's corruption budget during the run of a refresh protocol. Finally, the assumed mobility of the adversary often, somehow conveniently, aligns with the round complexity of computing a single layer of the layered circuit.

We do not make such assumptions about the maximally mobile adversary and, as such, designing secure protocols for maximally proactive MPC is significantly more challenging.

Definition 4 (Maximally Proactive MPC). *If Π is a protocol that securely (with erasures) computes any functionality f while executing in the presence of a maximally mobile adversary. Then, Π is a protocol for maximally proactive MPC.*

A protocol for maximally proactive secret sharing is an instance of maximally proactive MPC that allows a dealer to share a secret s among a group of n parties such that the secret remains secure against a maximally mobile adversary and allows the final shareholder set of n parties to open the secret. A refresh protocol prevents the adversary from discovering and destroying the secrets.

Definition 5 (Maximally Proactive Secret Sharing). *A maximally proactive secret sharing protocol is a set of instances of maximally proactive MPC, each associated with a* phase *and executing algorithms of a robust secret sharing scheme S. The initial phase is the* Share *phase, then a sequence of* Refresh *phases are executed, and finally the* Open *phase. Each phase is described below.*

- Share. *The instance of maximally proactive MPC for the* Share *phase has a designated input client P_D (the dealer) giving secret s as input and output clients $\mathsf{P}_1, \ldots, \mathsf{P}_n$ receiving shares $s_1, \ldots, s_n$ as output. In this phase, the maximally proactive MPC instance executes the* Sh *algorithm of the robust secret sharing scheme on input s to obtain $s_1, \ldots, s_n$.*
- Refresh. *In the refresh phase maximally proactive MPC takes as input shares $s_1, \ldots, s_n$ from the parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and outputs new and independent shares $\hat{s}_1, \ldots, \hat{s}_n$ to the same parties such that if $\mathsf{Rec}(\{s_i\}_{i\in[n]}) = s$, then $\mathsf{Rec}(\{\hat{s}_i\}_{i\in[n]}) = s$. That is, the MPC executes $\mathsf{Sh}(\mathsf{Rec}(\{s_i\}_{i\in[n]}))$ of S under fresh randomness and securely distributes the resulting shares to $\mathsf{P}_1, \ldots, \mathsf{P}_n$.*
- Open. *The final phase involves all n parties broadcasting their shares. Then, all honest parties run the reconstruction algorithm* Rec *of the underlying scheme S on the set of shares.*

2.3 Adaptivity and Composability in Layered MPC

Let Π_g be a layered protocol realizing functionality g with standalone t-security, and let Π_f be another layered protocol in which Π_g is used as a subroutine to implement g. Suppose the layers where g is computed using Π_g do not execute any other protocol in parallel; i.e., only a single invocation of Π_g is made in such layers. Then, to prove the security of Π_f, it is sufficient to show that Π_f is t-secure in the so called g-hybrid model, where the calls to the sub routine Π_g is replaced with calls to the functionality g itself. This allows for a modular construction and analysis of protocols.

Formally, the g-hybrid model involves a communication protocol as well as calls to functionality g. Suppose l is the designated output layer of g. In a protocol Π_f in g-hybrid model, parties in layer $i-1$ can send their inputs to functionality g in round i. The functionality will deliver the output of g to receivers in the output layer l in round l which may be used by the parties in executing Π_f.

The following proposition adapts the sequential composability theorem of [Can00] to the layered setting. The proposition holds simply because a layered protocol with d layers and n parties per layer is essentially a nd party protocol with communication between a pair of adjacent layers in every round.

Proposition 1 (Sequential Composability for layered protocols). *Suppose a (n, t, d)-layered protocol Θ implements a functionality g with perfect standalone t-security [Can00, Gol09]. Suppose a layered protocol Π with input layer $\mathcal{L}_0$ and output layer $\mathcal{L}'_d, d' > d$ invokes Θ as a subroutine from $\mathcal{L}_a$ to $\mathcal{L}_{a+d}$, where $0 \leq a < a + d \leq d'$. Π making subroutine calls to Θ is t-secure if it is t-secure in the g-hybrid model.*

Universal Composability. As discussed in Definition 1, 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 $\mathcal{L}_d$) of a layered network. We develop a protocol for computing general functionalities in the stand-alone model showing perfect security by means of a straight-line black-box simulator and, thus, we can invoke Theorem 1.2 in [KLR10] and argue that the protocol is, in fact, secure under the definition of universal composability[7].

On Adaptive Adversaries. In Definition 1, we define layered MPC in the presence of a *rushing* and *adaptive* adversary. Clearly, this extra power for the adversary separates layered MPC from maximally proactive MPC (Definition 3) and shows that layered MPC is strictly stronger. Looking forward, we will, however, only analyze the layered protocols with respect to static (and rushing) adversaries. To argue adaptive security, we need to be able to simulate even when the real world adversary corrupts a party midway through the protocol. [CDD$^+$04] showed an exotic example of a perfectly secure protocol with static security against malicious adversaries but without adaptive security. Fortunately, all our protocols are based on linear secret sharing which makes extending our analysis to layered (*and* adaptive) MPC significantly easier.

As an example, consider a simulator's job when a set of parties $\mathcal{C}$ is already corrupted during a protocol execution and a new party P_i has just been added to this set. First, the simulator needs to construct a complete view (including the input) of the honest P_i that is consistent with all messages exchanged with the ideal functionality and communication with parties in $\mathcal{C}$. Secondly, the simulator's state needs to be "extended" with this new information. Concretely, the state should be as if P_i has been corrupted from the start of the protocol but behaved honestly until this point. In our protocols for perfect layered MPC, we let the simulator handle this challenge using conditional sampling. Since parties in $\mathcal{C}$ will only hold shares of a linear secret sharing scheme, even if the newly corrupted P_i is the dealer of such shares we can simulate the randomness used in the sharing algorithm. This is feasible since as long as the shares of $n - t$ honest parties are fixing the secret, the simulator is free to change the randomness to be consistent with the shares of parties in $\mathcal{C}$. Finally we note that when referring to computationally secure (PRG-based) protocols, we either need to settle for non-adaptive security or implement the PRG in the random oracle model.

[7] While we can meaningfully argue that the final protocol for computing general functionalities is UC-secure, we do not treat individual components of this protocol in a UC manner. This would require a significant modelling effort of communication and synchronization for layered MPC and would be counterproductive in our effort to present layered MPC as a simple special case of secure MPC as in [Can00, Gol09].

3 Basic Primitives

We introduce the basic primitives Future Messaging (f_{FM}) and Multiparty Addition (f_{add}) that that serve as building blocks for later constructions. In the layered model, Future Messaging is a primitive which allows an input client S to securely send a message m to an output client R in a later layer. Multiparty Addition allows a subset of parties in a layer to broadcast the sum of their inputs to all parties in a later layer.

3.1 Future Messaging

Future Messaging emulates a *secure channel* between a sender S and a receiver R in a future layer. As such, the primitive is similar[8] to Secure Message Transmission (SMT) over the specific directed and layered network where intermediate nodes may take part in the protocol and not merely forward messages from adjacent nodes. The functionality is presented in Functionality 3.1.

Figure 3.1 (Future Messaging functionality f_{FM})

Public parameters: Sender $\mathsf{S} \in \mathcal{L}_0$, receiver $\mathsf{R} \in \mathcal{L}_d$ for $d > 0$ and message domain M.
Secret inputs: S has input $m \in M$.

f_{FM} receives m from S, and delivers m to R.

Parallel Composition. Functionality f_{FM} delivers a message from a sender to a receiver in a later layer $\mathcal{L}_d$. However, when our protocol implementing f_{FM} is composed in parallel, the resulting functionality is not the natural parallel composition of f_{FM} which takes the input from each sender to each receiver and delivers them.

In fact, this functionality is impossible to realize even in the trivial case of messaging from one layer to the very next using the provided secure communication link. As an example, suppose communication from $\mathsf{S}_1 \in \mathcal{L}_0$ to $\mathsf{R}_1 \in \mathcal{L}_1$ and from $\mathsf{S}_2 \in \mathcal{L}_0$ to $\mathsf{R}_2 \in \mathcal{L}_1$ are composed in parallel. Now, a rushing adversary corrupting S_1 and R_2 can collect the message from S_2 to R_2 and set this as the message from S_1 to R_1. Interestingly, this limitation persists when parallely composing our protocol for realizing f_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$ (even for $d > 1$) with t-security for $t < n/3$. See Remark 2 in Appendix B for more details.

We capture the functionality realized by parallel execution of our future messaging protocol using a corruption aware functionality in Figure 3.2.

A Protocol for Future Messaging. Realizing Future Messaging from a sender in $\mathcal{L}_0$ to a receiver in $\mathcal{L}_1$ is trivial since there is a secure communication link between any such pair.

A $(n, t, 2)$-layered protocol for Future Messaging from a sender in $\mathcal{L}_0$ to a receiver in $\mathcal{L}_2$ can be achieved as follows. Sender $\mathsf{S} \in \mathcal{L}_0$ shares the message m among the parties in $\mathcal{L}_1$ using a t-secure robust secret sharing scheme. The parties in $\mathcal{L}_1$ forward their shares to the receiver $\mathsf{R} \in \mathcal{L}_2$ who uses the reconstruction algorithm on the received shares to recover the message. By t-security of the secret sharing scheme, an adversary corrupting at most t parties in $\mathcal{L}_1$ learns nothing about the message. However, since the secret sharing scheme is t-robust, R correctly reconstructs m even if at most t corrupt parties send incorrect shares.

This idea can be generalized to construct Future Messaging from $\mathcal{L}_0$ to $\mathcal{L}_d$ for any $d > 2$ using the secure (n, t, ℓ)-layered protocol for Future Messaging from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ and then from $\mathcal{L}_\ell$ to $\mathcal{L}_d$. Here, ℓ is any number such that $0 < \ell < d$; specifically, we can take $\ell = \left\lfloor \frac{d}{2} \right\rfloor$. This is achieved as follows. The sender $\mathsf{S} \in \mathcal{L}_0$ produces shares $(s_1, \ldots, s_n)$ of its message m, and sends the share s_i to the i-th party (P_i^ℓ) in $\mathcal{L}_\ell$ using Future Messaging from $\mathcal{L}_0$ to $\mathcal{L}_\ell$. Each party in level ℓ forwards its share to the receiver using Future Messaging from $\mathcal{L}_\ell$ to $\mathcal{L}_d$.

[8] The instance of Future Messaging with honest sender in $\mathcal{L}_0$ and honest receiver in $\mathcal{L}_2$ is equivalent to perfect 1-way SMT.

Figure 3.2 (Corruption-aware parallel Future Messaging functionality f^n_{FM})

PUBLIC PARAMETERS: Senders $\mathsf{S}_1, \ldots, \mathsf{S}_n \in \mathcal{L}_0$, receivers $\mathsf{R}_1, \ldots, \mathsf{R}_n \in \mathcal{L}_d$ where $d > 0$. The domain $M_{i,j}$ of message from S_i to R_j.
SECRET INPUTS: Each S_i wants to send each R_j a message $m_{(i,j)} \in M_{i,j}$.
ADDITIONAL INPUT TO FUNCTIONALITY: Set of corrupted parties $\mathcal{I}_0 \subseteq \mathcal{L}_0$ and corrupted receivers $\mathcal{I}_d \subseteq \mathcal{L}_d$.

1. For each honest $\mathsf{S}_i \notin \mathcal{I}_0$ and each $\mathsf{R}_j \in \mathcal{L}_d$, f^n_{FM} receives message $m_{(i,j)}$ from S_i to R_j.
2. For each honest $\mathsf{S}_i \notin \mathcal{I}_0$ and corrupt $\mathsf{R}_j \in \mathcal{I}_d$, f^n_{FM} forwards $m_{(i,j)}$ to the (ideal) adversary.
3. For each corrupt $\mathsf{S}_i \in \mathcal{I}_0$ and each $\mathsf{R}_j \in \mathcal{L}_d$, f^n_{FM} receives from the (ideal) adversary the message $m_{(i,j)}$ that S_i wants to send to R_j.
4. For each $\mathsf{S}_i \in \mathcal{L}_0$ and $\mathsf{R}_j \in \mathcal{L}_d$, f^n_{FM} sends $m_{(i,j)}$ to R_j as message from S_i.

This protocol can be executed in parallel, for each sender in $\mathcal{L}_0$ and receiver in $\mathcal{L}_d$, in order to realize the corruption aware (parallel) functionality f^n_{FM} (Figure 3.2) from $\mathcal{L}_0$ to $\mathcal{L}_d$ using f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ and from $\mathcal{L}_\ell$ to $\mathcal{L}_d$. The protocol is formally described in Figure 3.3.

Lemma 2 (Layered protocol for f^n_{FM}). *Let* $(\mathsf{Sh}, \mathsf{Rec})$ *be a robust* (t, n) *secret-sharing scheme (Definition 6), the* (n, t, d)*-layered protocol in Figure 3.3 realizes the functionality* f^n_{FM} *in Figure 3.2 with perfect security for* $t < n/3$.

Proof. We formally describe the simulator and provide a formal proof in Appendix B.2.

Going forward, we will focus on the (non-parallel) Future Messaging functionality f_{FM} in Figure 3.1 from a designated sender in a layer to a designated receiver in a later layer. This is, indeed, a special case of f^n_{FM} ($n = 1$) and a protocol was outlined informally in the beginning of this section.

Theorem 6. *For any* $d > 0$*, and message domain* M*, there exists an* (n, t, d)*-layered protocol* Π_{FM} *that realizes* f_{FM} *from a sender in* $\mathcal{L}_0$ *to a receiver in* $\mathcal{L}_d$ *with communication complexity* $O(n^{\lceil \log d \rceil} \log |M|)$.

Proof. For $d = 1$, there is a trivial protocol that realizes f_{FM} in which the sender sends the message (from a domain M) directly to the receiver using the provided secure communication link. The communication complexity of realizing this is simply $\log |M|$.

Suppose $d > 1$ and $\ell = \lfloor \frac{d}{2} \rfloor$. Consider protocols Π and Π' that realize functionalities f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ and from $\mathcal{L}_\ell$ to $\mathcal{L}_d$, respectively for message domain M^n. In the protocol in Figure 3.3, the f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ and f^n_{FM} from $\mathcal{L}_\ell$ to $\mathcal{L}_d$ are called, sequentially. Hence, by the sequential modular composition theorem for layered protocols in Proposition 1, the protocol obtained by replacing these oracle calls with subroutine calls to Π and Π', is secure against any layered adversary that corrupts at most t parties in layers 1 to $\ell - 1$ and $\ell + 1$ to $d - 1$ in addition to corrupting at most t parties in layer ℓ. The communication complexity of the resulting protocol is the sum of communication complexity of Π and Π'. The statement of the theorem is obtained by recursion using this observation and the existence of the trivial protocol for realizing f_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_1$. □

Figure 3.3 (Π^n_{FM}, an (n, t, d)-layered protocol realizing f^n_{FM})

PUBLIC PARAMETERS: Senders $\mathsf{S}_1, \ldots, \mathsf{S}_n \in \mathcal{L}_0$, receivers $\mathsf{R}_1, \ldots, \mathsf{R}_n \in \mathcal{L}_d$ where $d > 1$.
SECRET INPUTS: Each S_i wants to send $m_{(i,j)} \in M$ to a each receiver R_j.
RESOURCES: f^n_{FM} (with message domain M^n) from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ and $\mathcal{L}_\ell$ to $\mathcal{L}_d$.

1. Each $\mathsf{S}_i, i \in [n]$ samples $(s_{(i,j),1}, \ldots, s_{(i,j),n}) \leftarrow \mathsf{Sh}(m_{(i,j)})$ for each $j \in [n]$.
2. For $k \in [n]$, S_i sets the message to $\mathsf{P}^\ell_k \in \mathcal{L}_\ell$ in f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ to $\big(s_{(i,1),k}, \ldots, s_{(i,n),k}\big)$.
3. Each party $\mathsf{P}^\ell_k : k \in [n]$ receives $(\hat{s}_{(i,1),k}, \ldots, \hat{s}_{(i,n),k})$ from $\mathsf{S}_i, i \in [n]$ (delivered by f^n_{FM}).
4. $\mathsf{P}^\ell_k, k \in [n]$ sets the message to $\mathsf{R}_j \in \mathcal{L}_d$ in f^n_{FM} from $\mathcal{L}_\ell$ to $\mathcal{L}_d$ to $(\hat{s}_{(1,j),k}, \ldots, \hat{s}_{(n,j),k})$.
5. Each receiver $\mathsf{R}_j : j \in [n]$ computes the message from $\mathsf{S}_i : i \in [n]$ as $\mathsf{Rec}(\hat{s}_{(i,j),1}, \ldots, \hat{s}_{(i,j),n})$.

Corollary 2. *Suppose Π_{FM} is a (n,t,d)-layered protocol realizing f_{FM} from a sender $\mathsf{S} \in \mathcal{L}_0$ to a receiver $\mathsf{R} \in \mathcal{L}_d$. The following statements hold when Π_{FM} is executed in the presence of any adversary $\mathcal{A}$ described in Definition 1:*

(a) If S is honest, R correctly recovers the input of S at the end of Π_{FM}.
(b) When S and R are honest, and for any pair of inputs $m, m' \in M$,

$$\mathrm{ADVR}_{\Pi_{\mathsf{FM}},\mathcal{A},\mathcal{I}}(m) \equiv \mathrm{ADVR}_{\Pi_{\mathsf{FM}},\mathcal{A},\mathcal{I}}(m').$$

3.2 Multiparty Addition

The Multiparty Addition functionality f_{add} takes inputs from a set of input clients and delivers the sum of the inputs to all output clients in $\mathcal{L}_2$. However, f_{add} allows the adversary to choose the inputs of corrupt input clients after learning the sum of the inputs of the honest clients. Hence, if at least one party with input to f_{add} is corrupt, the adversary can choose the value that f_{add} outputs. Note that, this necessarily makes f_{add} a corruption aware functionality. The functionality is formally defined in Figure 3.4.

Figure 3.4 (Multiparty Addition functionality f_{add})

PUBLIC PARAMETERS: Input clients $S \subseteq \mathcal{L}_0$, output clients $\mathcal{L}_2$, input domain is some finite group $\mathbb{G}$.
SECRET INPUTS: Each $\mathsf{S}_i \in S$ has input $x_i \in \mathbb{G}$.
ADDITIONAL INPUT: Set of corrupt input clients $\mathcal{I} \subseteq S$.

1. f_{add} receives input x_i from each honest $\mathsf{S}_i \in S$.
2. If $\mathcal{I} \neq \emptyset$, f_{add} leaks the sum of the inputs of honest S_i to the (ideal) adversary and receives a value $y \in \mathbb{G}$ from the adversary.
3. To all output clients in $\mathcal{L}_2$, f_{add} delivers y if $\mathcal{I} \neq \emptyset$ and $\sum_{i:\mathsf{S}_i \in S} x_i$ otherwise.

The functionality described in Figure 3.4 can be realized by an $(n,t,2)$-layered protocol in the following way. Each party in $\mathsf{S}_i \in S$ secret shares its input x_i to the parties in next layer using a t-robust linear secret sharing scheme. Parties in $\mathcal{L}_1$ broadcasts the sum of their respective shares for each of the inputs. Each party in $\mathcal{L}_2$ recovers the output by running the reconstruction algorithm on the received sum of shares. A formal description of the protocol is presented in Figure 3.5.

Figure 3.5 (Π_{add}, an $(n,t,2)$-layered protocol for f_{add})

PUBLIC PARAMETERS: Input clients $\mathsf{P}_i^0, i \in S$, output clients $\mathcal{L}_2$, input domain is some finite group $\mathbb{G}$.
Robust (n,t) secret-sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ with secret domain M.
SECRET INPUTS: Each $\mathsf{S}_i \in S$ has input $x_i \in \mathbb{G}$.

1. Each $\mathsf{P}_i^0, i \in S$ samples $(x_{i,1}, \ldots, x_{i,n}) \leftarrow \mathsf{Sh}(x_i)$, and sends $x_{i,j}$ to P_j^1 for each $j \in [n]$.
2. Each $\mathsf{P}_j^1, j \in [n]$ broadcasts $y_j = \sum_{i \in S} x_{i,j}$ to $\mathcal{L}_2$.
3. Each party outputs $\mathsf{Rec}(y_1, \ldots, y_j)$. We use a reconstruction function that outputs a valid element of $\mathbb{G}$ even when $y_1, \ldots, y_n$ is not a valid secret sharing with at most t corruptions.

Clearly, all honest parties output the same value at the end of the protocol, irrespective of the number of corruption in S. If all parties in S are honest, each party in $\mathcal{L}_2$ receives a share of $\sum_{\mathsf{S}_i \in S} x_i$ for each party in $\mathcal{L}_1$. Although corrupt parties in $\mathcal{L}_1$ can potentially send invalid shares, by t-robustness of the secret sharing scheme all honest parties in $\mathcal{L}_2$ correctly reconstruct the sum of the inputs. Finally, the adversary who corrupts a non-empty set of parties in $\mathcal{L}_2$ only learns the sum of the shares of the honest parties' inputs. Since the secret sharing scheme is linear, this would only reveal the sum of the honest parties' inputs.

The following lemma formally states the game based security guarantees of any (n,t,d)-layered protocol realizing Multiparty Addition as per above.

Lemma 3. *Let $t, n, d \in \mathbb{N}$ and suppose protocol Π_{add} is an (n, t, d)-layered protocol realizing f_{add}. The following statements hold when Π_{add} is executed in the presence of any adversary $\mathcal{A}$ described in Definition 1:*

1. *All honest clients output the same value at the end of Π_{add}. If all input clients are honest, this value coincides with the sum of the inputs.*
2. *The view of $\mathcal{A}$ only reveals the sum of the inputs of the honest parties.*

4 Layered MPC based on CNF Secret Sharing

In this section, we start by building a protocol for Future Multicast based on primitives from Section 3. The protocol is then used in a simple way to obtain VSS using CNF-shares. We will build on this VSS protocol in order to realize secure multiplication and, finally, a protocol for layered MPC for *any* function.

4.1 Future Multicast

Future Multicast f_{FMcast} allows a sender S to send a secret to a set of receivers R located in a later layer. It guarantees that all honest receivers output the same value even if the sender is corrupt; if the sender is honest, this value coincides with the sender's input. Finally, if all receivers (and the sender) are honest, the secret remains hidden from the adversary. This primitive will be the backbone of our layered VSS protocol. Standard (Secure) Multicast is often described as the simplest non-trivial example of secure computation. Also, in layered MPC, Future Multicast generalizes Future Messaging and Future Broadcast[9] but is substantially harder to realize. The functionality is described in Figure 4.1.

Figure 4.1 (Future Multicast functionality f_{FMcast})

Public parameters: Sender $\mathsf{S} \in \mathcal{L}_0$, receiving set of parties $R \subseteq \mathcal{L}_d, d \geq 5$, message domain M.
Secret inputs: S has input $m \in M$.

f_{FM} receives m from S, and delivers m to all parties in R.

A protocol for Future Multicast. As a first step towards realizing f_{FMcast}, we construct a protocol that achieves a weaker notion of Future Multicast. In this protocol, sender S in layer $\mathcal{L}_0$ sends a share to a set of intermediaries $U_T = \{\mathsf{P}_i^1 : i \in T\} \subset \mathcal{L}_1$, in the next layer, who communicate it to the receivers $R \subseteq \mathcal{L}_5$. The protocol for *weak Future Multicast* provides the following guarantees which are formally stated in Lemma 4.

1. (*Agreement*). If a majority of the intermediaries are honest, all honest receivers output the same value at the end of the protocol even if S is corrupt; if the sender is honest, this value coincides with the sender's input.
2. (*Security*). If the sender, all the intermediaries in U_T and all the receivers are honest, a layered adversary does not learn the sender's secret.

Observe that, when $t < n/3$, each subset U_T of $n - t$ parties in $\mathcal{L}_1$ contains a strict minority of corrupt parties. Furthermore, there is at least one such set that contains only honest parties. Given these observations, realizing f_{FMcast} from the weaker notion is straight forward: For each set $U_T \subset \mathcal{L}_1$ of $n - t$ parties, S sends r_T to the receivers using parties in U_T as intermediaries, where r_T for all possible T, form an additive secret sharing of the sender's secret. When the sender is honest and each set of intermediaries has an honest majority, by (1), all r_T reach the receivers correctly. Furthermore, for one set of intermediaries U_{T^*}, by (2), r_{T^*} remains hidden from the adversary. Thus, receivers can compute the sum of r_T for distinct sets T to obtain the secret, which will remain hidden from the

[9] Here, we refer to the primitive in the setting of layered MPC that ensures termination, validity and agreement among all parties located in some layer $d > 1$. Not Future Broadcast as defined in [GHK+21].

layered adversary if all receivers are honest. Finally, by (2), the outputs of all honest receivers are consistent even if the sender is corrupt.

Weak Future Multicast. With the aid of a set of intermediaries $U_T = \{\mathsf{P}_i^1 : i \in T\} \subset \mathcal{L}_1$, weak Future Multicast can be achieved as follows: S sends the message r_T to every party in U_T. In addition, S distributes a robust secret sharing of r_T among the parties in $\mathcal{L}_3$ using Future Messaging. Every pair of intermediaries broadcasts the difference between the values they received to all parties in $\mathcal{L}_3$ using a protocol for the f_{add} functionality. Additionally, each intermediary distributes a secret sharing of the value they received among the parties in $\mathcal{L}_4$. If the difference comes out non-zero for any pair, the parties in $\mathcal{L}_3$ effectively reveals r_T to all parties in $\mathcal{L}_4$ by broadcasting the shares of r_T that S distributed. Parties in $\mathcal{L}_4$ then forwards (using layer-to-layer broadcast) r_T to all the receivers in R. By robustness of the secret sharing scheme, parties in $\mathcal{L}_4$ recover r_T if it was secret shared properly by the sender; moreover, even if S sent invalid shares, all honest parties recover the same value. Hence, receivers recover r_T from this because at most $t < n/3$ parties in $\mathcal{L}_4$ are corrupt. If the difference is zero for every pair of intermediaries, each party in $\mathcal{L}_4$ reveals the share sent to them by every intermediary to all the receiver in R. Using these shares, each receiver reconstructs the value that was shared by each intermediary. If the difference was zero for every pair of intermediaries, then all honest intermediaries must have received the same value from S (which is r_T if S is honest). Hence, a majority of the values recovered by every receiver coincides with this value. This ensures (1). If S and all intermediaries are honest, r_T is not revealed to parties in $\mathcal{L}_4$, and, hence, is disclosed only to the receivers ensuring (2).

An $(n, t, 5)$-layered protocol for Future Multicast Π_{FMcast} is formally described in Figure 4.2. Importantly, it includes the sub-protocol for weak Future Multicast $\Pi_{\mathsf{weak\text{-}FMcast}}$. We identify two important properties of Π_{FMcast} that will be used going forward. The properties are stated in Lemma 4 and a formal proof is provided in Appendix C.1.

Lemma 4. *For any $T \in \mathcal{T}$, the following properties hold for $\Pi_{\mathsf{weak\text{-}FMcast}}$ with U_T as intermediaries when executed in the presence of any adversary $\mathcal{A}$:*

(a) There exists $\hat{r}$ such that all honest receivers in R output $\hat{r}$ at the end of $\Pi_{\mathsf{weak\text{-}FMcast}}$. Furthermore, if S is honest, $\hat{r} = r$.

(b) If S, and all intermediaries and receivers are honest, for any $r, r' \in M$,

$$\mathrm{ADVR}_{\Pi,\mathcal{A}}(r) \equiv \mathrm{ADVR}_{\Pi,\mathcal{A}}(r').$$

Theorem 7. *Protocol Π_{FMcast} in Figure 4.2 is a secure $(n, t, 5)$-layered protocol realizing f_{FMcast} with input client S and output clients in R.*

Proof. By statement (a) in Lemma 4, for every set of intermediaries $\{\mathsf{P}_i^1 : i \in T\}$, there exists $\hat{r}_T$ such that all honest receivers in R output $\hat{r}_T$ at the end of $\Pi_{\mathsf{weak\text{-}FMcast}}$. Furthermore, if S is honest, $\hat{r}_T = r_T$, for each $T \in \mathcal{T}$. Hence, the outputs of all receivers are the same at the end of Π_{FMcast} and coincides with the input of an honest S.

It remains to show that if the sender and all receivers are honest, $\mathcal{A}$ does not learn the sender's input. We sketch the intuition: Consider $T^* \in \mathcal{T}$ such that the parties U_{T^*} are all honest; such a set exists because there are at most t corruptions in each layer. By statement (b) in Lemma 4, view of $\mathcal{A}$ interacting with $\Pi_{\mathsf{weak\text{-}FMcast}}$ with intermediaries in U_{T^*} is independent of the input r_{T^*} of S. But then, the view of $\mathcal{A}$ in the entire protocol Π_{FMcast} does not depend on m since $(r_T, T \in \mathcal{T})$ is an additive secret sharing of m. We formally prove security of Π_{FMcast} by demonstrating a simulator $\mathcal{S}$ in Appendix C.2.

Figure 4.2 (Π_{FMcast}, an $(n, t, 5)$-layered protocol for f_{FMcast})

PUBLIC PARAMETERS: Sender $\mathsf{S} \in \mathcal{L}_0$, receivers $R \subseteq \mathcal{L}_d$, where $d = 5$,
A (t, n) robust linear secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$.
DEFINITIONS: $\mathcal{T} = \{T \subset [n] : |T| = n - t\}$ (Definition 7).
SECRET INPUTS: S has input $m \in M$.
SUBROUTINES: Protocol Π_{FM} realizing f_{FM} and Π_{add} realizing f_{add}.

1. S samples $\{r_T\}_{T\in\mathcal{T}}$ uniformly at random conditioned on $m = \sum_{T\in\mathcal{T}} r_T$. For each $T \in \mathcal{T}$, execute protocol $\Pi_{\mathsf{weak\text{-}FMcast}}$ (described below) with $U_T = \{\mathsf{P}^1_i : i \in T\}$ as intermediaries and r_T as input from S.
2. For each $T \in \mathcal{T}$, suppose $\hat{r}_T$ is the output of receiver $\mathsf{P}^5_i \in R$ at the end of $\Pi_{\mathsf{weak\text{-}FMcast}}$ with U_T as intermediaries. P^5_i outputs $\hat{m} = \sum_{T\in\mathcal{T}} \hat{r}_T$.

SUB-PROTOCOL: $\Pi_{\mathsf{weak\text{-}FMcast}}$ with public input U_T and r as input from S.

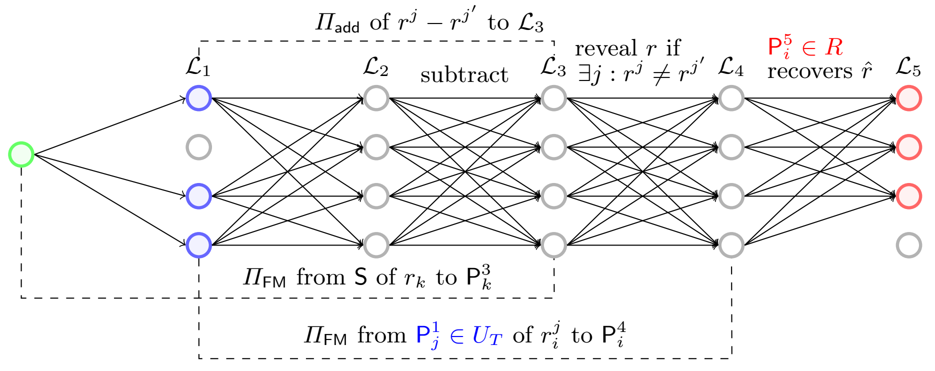
(i). Layer 0**:**
1. S sends r to parties in $U_T \subset \mathcal{L}_1$ over the secure channel.
2. S samples $(r_1, \ldots, r_n) \leftarrow \mathsf{Sh}(r)$. For $k \in [n]$, S sends r_k to P^3_k using Π_{FM}.

(ii). Layer 1**:**
1. Denote the value received by $\mathsf{P}^1_j \in U_T$ by r^j. For each $j, j' \in T$ such that $j < j'$, execute Π_{add} to compute $r^j - r^{j'}$ and broadcast the result to $\mathcal{L}_3$.
2. Each intermediary $\mathsf{P}^1_j \in U_T$ samples $(r^j_1, \ldots, r^j_n) \leftarrow \mathsf{Sh}(r^j)$ and sends r^j_i to $\mathsf{P}^4_i, i \in [n]$ using Π_{FM}.

(iii). Layer 3**:**
1. Each party $\mathsf{P}^3_k, k \in [n]$ recovers $\hat{r}_k$ as the output of Π_{FM} (see step (i).2). P^3_k broadcasts a complaint and $\hat{r}_k$ to all parties in $\mathcal{L}_4$ if $r^j - r^{j'} \neq 0$ (output of Π_{add}) for some j, j'.

(iv). Layer 4**:**
1. If at least $n - t$ parties in $\mathcal{L}_3$ broadcasted a complaint, each $\mathsf{P}^4_i, i \in [n]$ computes $\hat{r} = \mathsf{Rec}(\hat{r}_1, \ldots, \hat{r}_n)$. P^4_i forwards the complaint to all receivers in R and sends $\hat{r}$ to all receivers.
2. Else, P^4_i recovers $\hat{r}^j_i$ as the output of Π_{FM} (see step (ii).2) with P^1_j as sender (with message r^j_i) and sends it to all receivers.

(v). Layer 5**:**
1. If at least $n - t$ parties in $\mathcal{L}_4$ reported a complaint, each $\mathsf{P}^5_i \in R$ outputs the unique value that is sent by at least $n - t$ parties in $\mathcal{L}_4$.
2. Else, P^5_i recovers $\hat{r}^j = \mathsf{Rec}(\hat{r}^j_1, \ldots, \hat{r}^j_n)$, where $\hat{r}^j_i$ was sent by P^4_i and outputs the unique value $\hat{r}$ such that $\hat{r} = \hat{r}^j$ for at least $n - 2t$ distinct values of $j \in T$.



Π_{add} of $r^j - r^{j'}$ to $\mathcal{L}_3$
$\mathcal{L}_1$
$\mathcal{L}_2$
subtract
$\mathcal{L}_3$
reveal r if $\exists j : r^j \neq r^{j'}$
$\mathcal{L}_4$
$\mathsf{P}^5_i \in R$ recovers $\hat{r}$
$\mathcal{L}_5$
Π_{FM} from S of r_k to P^3_k
Π_{FM} from $\mathsf{P}^1_j \in U_T$ of r^j_i to P^4_i

Fig. 2. Execution flow of $\Pi_{\mathsf{weak\text{-}FMcast}}$ in Protocol 4.2

4.2 Verifiable Secret Sharing

Using the FMcast primitive presented in Section 4.1, realizing verifiable secret sharing (VSS) is relatively straight-forward. The sender distributes the additive shares of the secret to each set of receivers using Future Multicast. The protocol in Figure 4.3 realizes VSS from a dealer in $\mathcal{L}_0$ to shareholders in $\mathcal{L}_5$.

Figure 4.3 (Π_{VSS}, an $(n,t,5)$-layered protocol for f_{VSS})

PUBLIC PARAMETERS: Sender $\mathsf{S} \in \mathcal{L}_0$, shareholders $\mathcal{L}_5$.
DEFINITIONS: Let $\mathcal{T} = \{T \subset [n] : |T| = n - t\}$.
SECRET INPUTS: S has input $m \in M$.
SUBROUTINES: Protocol Π_{FMcast} realizing f_{FMcast} functionality.

Layer $\mathcal{L}_0$:
1. S samples $(r_T)_{T\in\mathcal{T}}$ as additive secret sharing of m.
2. For each $T \in \mathcal{T}$, execute Π_{FMcast} with S as sender with input r_T and $\{\mathsf{P}_i^5 : i \in T\}$ as receivers.

Layer $\mathcal{L}_5$:
1. Each party $\mathsf{P}_i^5, i \in [n]$ recovers r_T as the output of Π_{FMcast} with S as sender if $i \in T$. P_i^5 outputs $(r_T)_{i\in T}$ as its share.

The protocol described in Figure 4.3 is a $(n,t,5)$-layered protocol realizing VSS. This follows from the definition of Future Multicast. The following theorem proves a stronger result: Suppose n protocols are executed in parallel with P_i^0 as dealer and $\mathcal{L}_5$ as shareholders for each $i \in [n]$, then we achieve a parallel $(n,t,5)$-layered protocol for VSS functionality for $t < n/3$. The parallel VSS fuctionality is formally described below.

Figure 4.4 (Parallel VSS functionality $f_{\mathsf{parallel\text{-}VSS}}$)

PUBLIC PARAMETERS: Senders $\mathsf{S}_1, \ldots, \mathsf{S}_n \in \mathcal{L}_0$, shareholders $\mathsf{R}_1, \ldots, \mathsf{R}_n \in \mathcal{L}_5$.
The domain M of secrets.
DEFINITIONS: Let $\mathcal{T} = \{T \subset [n] : |T| = n - t\}$.

1. Each $\mathsf{S}_i, i \in [n]$ sends $(r_T^i)_{T\in\mathcal{T}}$ to the functionality.
2. For each $i \in [n]$ and $T \in \mathcal{T}$, functionality sends (i, T, r_T^i) to $\{\mathsf{P}_j^5 : j \in T\}$.

Theorem 8. *The protocol in Figure 4.3 executed in parallel realizes $f_{\mathsf{parallel\text{-}VSS}}$ with perfect t-security for $t < n/3$ by consuming 5 layers, and by communicating $\binom{n}{t}^3 \cdot O(n^2)$ field elements over the point-to-point channels and over the broadcast channels for each secret.*

Proof. The VSS protocol is essentially several multicast protocols executed in parallel. The security of the construction follows from the security of the multicast protocol, once we ensure that the adversary cannot correlate the shares of the corrupt parties with those of the honest parties across parallel executions of multicast protocols. The simulator for multicast extracts the input of a corrupt sender in $\mathcal{L}_0$ from the view of the honest parties in the protocol up to $\mathcal{L}_4$. This allows the simulator we build for parallel VSS to extract the shares of the corrupt dealers after simulating the protocol till $\mathcal{L}_4$ and provide them to $f_{\mathsf{parallel\text{-}VSS}}$. Whereas, a multicast from an honest sender to a set of receivers, potentially containing corrupt receivers, does not reveal the sent message to the corrupt parties until $\mathcal{L}_4$. Hence, the adversary chooses shares for corrupt parties before getting to see the shares chosen by the honest parties. This guarantees that the adversary cannot correlate the shares of the corrupted parties with the shares of the honest parties. We show a simulator and full proof in Appendix C.3. □

Addition and multiplication-by-constant for CNF shares. The CNF secret sharing scheme is linear; hence, parties holding valid CNF shares of a value s can locally transform it into a valid secret sharing of αs when α is a publicly known constant. In detail, let s_i be the share of s held by party i. Then, there exist $(\delta_T)_{T\in\mathcal{T}}$ such that $\sum_{T\in\mathcal{T}} \delta_T = s$, and $s_i = (\delta_T)_{T:i\in T}$ for each $i \in [n]$. Then $s'_1, \ldots, s'_n$ such that $s'_i = (\alpha\delta_T)_{T:i\in T}$ is a CNF secret sharing of αs. Additionally, suppose a value

r is secret shared as $(r_1, \ldots, r_n)$ where $r_i = (\gamma_T)_{T:i\in T}$ for each $i \in [n]$, and $\sum_{T\in\mathcal{T}} \gamma_T = r$. Then, $s''_1, \ldots, s''_n$ such that $s''_i = (\delta_T + \gamma_T)_{T:i\in T}$ is a CNF secret sharing of $r + s$. In conclusion, addition and multiplication by constant of CNF shares can be computed locally.

4.3 Multiplication

The multiplication functionality takes valid CNF secret shares of two values r and s and computes a fresh CNF secret shares of rs. This functionality requires that the input clients hold valid CNF secret sharing of the individual values to be multiplied, and that at most t input clients are corrupt. Recall that, a default layered adversary is allowed to corrupt arbitrarily many input and output clients. The functionality is formally defined as follows:

Implementing f_{mult}. Suppose $r_1, \ldots, r_n$ and $s_1, \ldots, s_n$ are CNF secret shares of two values r and s, respectively. Recall that, when $\mathcal{T} = \{T_1, \ldots, T_N\} = \{T \subset [n] : |T| = n - t\}$, for each $i \in [n]$, $r_i = (\gamma_j)_{j:i\in T_j}$ and $s_i = (\lambda_j)_{j:i\in T_j}$, where $\sum_{i=1}^{N} \gamma_j = r$ and $\sum_{i=1}^{N} \lambda_j = s$. To compute a secret sharing of rs, it suffices to compute the secret sharing of $\gamma_i\lambda_j$ for every $i, j \in [N]$; secret shares of rs can be computed as the sum of these secret shares, which can be obtained by local computations. This follows from the fact that, $rs = \sum_{i=1}^{N}\sum_{j=1}^{N} \gamma_i\lambda_j$.

The main challenge in implementing multiplication is in obtaining correct secret shares of $\gamma_i\lambda_j$, for all $i, j \in [N]$. In the non-layered setting, classic protocols tackle this by having all parties who have access to γ_i and λ_j secret share their product. The parties then compute the difference between the values shared as purported product $\gamma_i\lambda_j$ by securely computing their differences. If all differences come out to be 0, since at least one of the parties secret sharing the product is honest, all the remaining parties must also have correctly shared the secret. Hence, one of these CNF-shares can be taken as a valid secret sharing of $\lambda_i\gamma_j$. Whenever the difference is non-zero, both γ_i and λ_j are publicly revealed, and a trivial secret sharing of $\gamma_i\lambda_j$ is taken instead of the ones submitted by the parties. Finally, these shares are 'added' together to get a secret sharing of rs.

Figure 4.5 (Multiplication functionality f_{mult})

Public parameters: Input layer $\mathcal{L}_0$, output layer $\mathcal{L}_7$.
Secret inputs: Each $\mathsf{P}^0_i, i \in [n]$ receives (r_i, s_i), where $(r_i)_{i\in[n]}$ and $(s_i)_{i\in[n]}$ are valid CNF secret sharing of r and s, respectively.

1. Each party $\mathsf{P}^0_i, i \in [n]$ sends (r_i, s_i) to f_{mult}, who reconstructs r from $(r_i)_{i\in[n]}$ and s from $(s_i)_{i\in[n]}$. This is possible since at most t parties are corrupt and CNF secret sharing is t robust.
2. f_{mult} samples $(u_1, \ldots, u_n)$ as a fresh CNF secret sharing of rs. For each $i \in [n]$, f_{mult} delivers u_i to $\mathsf{P}^7_i, i \in [n]$ in the output layer.

The above protocol is clearly correct. The security of the protocol follows from the fact that, whenever all the parties submitting shares of $\gamma_i\lambda_j$ for some i, j are honest, the protocol never reaches the public reveal phase. A formal description of the protocol in the standard setting as constructed in [Mau06] is provided in Figure C.1 in Appendix C.4. Our multiplication protocol is a porting of the above protocol to the layered setting. In the process, we face two main challenges.

Firstly, when the public check of equality between purported shares of $\gamma_j \cdot \lambda_{j'}$ provided by a pair of parties fails in step 2, γ_j and $\lambda_{j'}$ need to be revealed by every party (in the input layer) with access to these values. This is tackled exactly as in the Future Multicast protocol. Using Future Messaging, all parties in the input layer secret share each γ_i and λ_i they hold to the layer where the equality check is made; the parties in this layer then selectively reveal the additive shares for which any of the equality checks fails.

The second challenge is less straightforward to handle. If the protocol is naively ported to the layered model, VSS of $\gamma_j \cdot \lambda_{j'}$ will be available in two different layers: once in the layer that initiates the equality check, and then again in the final layer that computes the VSS of $r \cdot t$ as the sum of VSS of $\gamma_j \cdot \lambda_{j'}$ for all $j, j' \in [N]$. But then, the adversary can corrupt t parties in both these layers, and recover $\gamma_j \cdot \lambda_{j'}$ for each (j, j'). This is overcome as follows: For each j, j', consider the special party whose share of $\gamma_j \cdot \lambda_{j'}$ will be chosen in the final addition (if the all equality checks for $\gamma_j \cdot \lambda_{j'}$ succeeds). This party samples $(\delta_k)_{k\in[N]}$ as additive secret shares of $\gamma_j\lambda_{j'}$, and verifiable secret share each δ_k instead of directly secret sharing $\gamma_j \cdot \lambda_{j'}$. The equality check is now carried out to check if $\sum_k \delta_k$ shared by the special party equals the value shared by every other party. Finally, parties in the output layer receive a VSS of $\gamma_j \cdot \lambda_{j'}$ in which the ith share is $(\delta_k)_{k:i\in T_k}$. This avoids reuse of the same VSS in two layers. The protocol is presented in Figure 4.6.

Figure 4.6 (Π_{mult}, an $(n, t, 7)$-layered protocol for realizing f_{mult})

PUBLIC PARAMETERS: Input layer $\mathcal{L}_0$, output layer $\mathcal{L}_7$. A (t, n) robust perfectly secure secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$

INPUTS: Party $\mathsf{P}_i^0, i \in [n]$ has inputs (r_i, s_i), where $(r_i)_{i\in[n]}$ and $(s_i)_{i\in[n]}$ are CNF secret shares of r and s, respectively.

DEFINITIONS: Let $\mathcal{T} = \{T_1, \ldots, T_N\} = \{T \subset [n] : |T| = n - t\}$.
For each $i \in [n]$, $s_i = (\gamma_j : i \in T_j, j \in [N])$
and $r_i = (\lambda_j : i \in T_j, j \in [N])$, where $\sum_{j\in[N]} \gamma_j = r$ and $\sum_{j\in[N]} \lambda_j = s$.

SUBROUTINES: Protocols Π_{FM}, Π_{VSS}.

1. For each $j, j' \in [N]$, concurrently execute protocol $\Pi_{j,j'}$ (described below).
2. For each $j, j' \in [N]$, let $(\delta_k^{j,j'})_{k:i\in T_k}$ be the output of party $\mathsf{P}_i^7, i \in [n]$ at the end of $\Pi_{j,j'}$. Then, P_i^7 stores $(\delta_k)_{k:i\in T_k}$ as their shares of rs, where $\delta_k = \sum_{j,j'\in[N]} \delta_k^{j,j'}$.

SUB-PROTOCOL: $\Pi_{j,j'}$ for $j, j' \in [N]$

NOTATIONS: Let $I = T_j \cap T_{j'}$; i.e., P_i^0 has both γ_j and $\lambda_{j'}$ if and only if $i \in I$. Fix $i^* \in I$. To avoid redundancy, denote γ_j by γ and $\lambda_{j'}$ by λ.

(i). Layer $\mathcal{L}_0$:

1. $\mathsf{P}_{i^*}^0$ samples $\delta_1, \ldots, \delta_N$ as additive secret shares of $\gamma\lambda$. For each $k \in [N]$, execute Π_{VSS} with $\mathsf{P}_{i^*}^0$ as dealer with input δ_k and $\mathcal{L}_5$ as shareholders. For each $i \in I, i \neq i^*$, execute Π_{VSS} with P_i^0 as dealer with input $\gamma\lambda$ and $\mathcal{L}_5$ as shareholders.
2. Each $\mathsf{P}_i^0, i \in T_j$ samples $(\gamma^{i,1}, \ldots, \gamma^{i,n}) \leftarrow \mathsf{Sh}(\gamma)$. For each $k \in [n]$, execute Π_{FM} with P_i^0 as sender with input $\gamma^{i,k}$ and P_k^6 as receiver.
3. Similarly, Each $\mathsf{P}_i^0, i \in T_{j'}$ samples $(\lambda^{i,1}, \ldots, \lambda^{i,n}) \leftarrow \mathsf{Sh}(\lambda)$. For each $k \in [n]$, execute Π_{FM} with P_i^0 as sender with input $\lambda^{i,k}$ and P_k^6 as receiver.

(ii). Layer $\mathcal{L}_5$:

1. For each $i \in I, i \neq i^*$, we will denote the value verifiably secret shared by P_i^0 in step (i).1 by $(\gamma\lambda)^{(i)}$. If P_i^0 is honest, $(\gamma\delta)^{(i)} = \gamma\lambda$. Parties in $\mathcal{L}_5$ locally compute that shares of $(\gamma\lambda)^{(i)} - \sum_{l=1}^{N} \delta_l$ and broadcast the shares of the sum to $\mathcal{L}_6$.
2. For each $k \in [N]$, $\mathsf{P}_l^5, l \in [n]$ reveals their share of δ_k to each party $\mathsf{P}_i^7, i \in T_k$ using Future Messaging Π_{FM}.

(iii). Layer $\mathcal{L}_6$:

1. Each $\mathsf{P}_k^6, k \in [n]$ recovers $\gamma^{i,k}$ as the output of Π_{FM} with $\mathsf{P}_i^0, i \in T_j$ as sender and P_k^6 as receiver initiated in step (i).2. Similarly, P_k^6 recovers $\lambda^{i,k}$ as the output of Π_{FM} with $\mathsf{P}_i^0, i \in T_{j'}$ as sender and P_k^6 as receiver initiated in step (i).3.
2. If there exists $i \neq i^*$ such that $(\gamma\lambda)^{(i)} - \sum_{l=1}^{N} \delta_l \neq 0$, then P_k^6 sends a complaint along with $(\gamma^{i,k})_{i\in T_j}$ and $(\lambda^{i,k})_{i\in T_{j'}}$ to all parties in the next layer.

(iv). Layer $\mathcal{L}_7$:

1. If at least $n - t$ complaints are received, $\mathsf{P}_l^7, l \in [n]$ recovers $\gamma^i = \mathsf{Rec}(\gamma^{i,1}, \ldots, \gamma^{i,n})$ for each $i \in T_j$, and recovers $\lambda^i = \mathsf{Rec}(\lambda^{i,1}, \ldots, \lambda^{i,n})$ for each $i \in T_{j'}$. Set γ as the unique value such that $\gamma = \gamma^i$ for at least $n - 2t$ distinct $i \in T_j$; similarly, set λ as the unique value such that $\lambda = \lambda^i$ for at least $n - 2t$ distinct $i \in T_{j'}$. Finally, P_l^7 sets the share $(rs)_l^{j,j'}$ to be the trivial sharing of $\gamma\lambda$; i.e., $(rs)_l^{j,j'} = (\delta_k)_{k:l\in T_k}$, where $\delta_k = \gamma\lambda$, if $k = 1$ and $\delta_k = 0$ otherwise.
2. If less than t complaints were received, each party $\mathsf{P}_i^7, i \in [n]$ recovers the CNF share of δ_k (for each k such that $i \in T_k$) sent by $\mathsf{P}_l^5, l \in [n]$ using Π_{FM} in step (ii).2. Using these shares, P_i^7 recovers δ_k and sets the share $(rs)_i^{j,j'}$ to be $(\delta_k)_{k:i\in T_k}$.

We first establish properties of the subroutine $\Pi_{j,j'}$ that computes CNF shares of $\gamma_j \cdot \lambda_{j'}$ for each $j, j' \in [N]$. in the lemma below, proven in Appendix C.5.

Lemma 5. *For any $j, j' \in [N]$, the following properties hold for $\Pi_{j,j'}$ when executed in the presence of an adversary $\mathcal{A}$:*

(a) There exists $(\delta_k)_{k\in[N]}$ such that $\sum_{k=1}^{N} \delta_k = \lambda_j\gamma_{j'}$, and each honest party $\mathsf{P}_i^7, i \in [N]$ outputs $(\delta_k)_{k:i\in T_k}$ at the end of $\Pi_{j,j'}$.
(b) Suppose parties $\mathsf{P}_i^0, i \in H$ are honest, then for any a, b, a', b',

$$\mathrm{ADVR}_{\Pi_{j,j'},\mathcal{A}}(\gamma_j = a, \lambda_{j'} = b) \equiv \mathrm{ADVR}_{\Pi_{j,j'},\mathcal{A}}(\gamma_j = a', \lambda_{j'} = b').$$

By statement (a) in Lemma 5, for each $j, j' \in [N]$, parties in the output layer correctly receive a CNF secret sharing of $\gamma_j\lambda_{j'}$. Hence, the output of the parties at the end of the protocol is a CNF secret sharing of $\sum_{j=1}^{N}\sum_{j'=1}^{N} \gamma_j\lambda_{j'} = rs$. By statement (b) in Lemma 5, if $\lambda_{j'}$ or γ_j is not known to the adversary, the output of $\Pi_{j,j'}$ does not reveal $\gamma_j\lambda_{j'}$. This ensures that the protocol is secure. The following theorem formally proves the security of the protocol.

Theorem 9. *Π_{mult} in Figure 4.6 is an $(n, t, 7)$-layered protocol realizing f_{mult}, for $t < n/3$.*

Proof. We prove the security by constructing a simulator $\mathcal{S}$. In each layer $\mathcal{L}_l, 0 \le l \le 7$, let $\mathsf{P}_i^l, i \in C_l$, where $C_l \subset [n]$, be the set of corrupt parties. For each corrupt party $\mathsf{P}_i^0, i \in C_l$, $\mathcal{S}$ receives $(\gamma_k)_{k:i\in T_k}$ and $(\lambda_k)_{k:i\in T_k}$ from the environment. $\mathcal{S}$ chooses γ_k arbitrarily for each k such that γ_k is not provided by the functionality. The simulator then emulates all the honest parties and interacts with the adversary. For this, $\mathcal{S}$ is required to set the input to the honest parties in $\mathcal{L}_0$ (these are the only parties with inputs). For each $i \notin C_0$, $\mathcal{S}$ sets the shares of r and s of honest emulated party P_i^0 to be $(\gamma_k)_{k:i\in T_k}$ and $(\lambda_k)_{k:i\in T_k}$, respectively.

For each pair j, j' such that γ_j and $\lambda_{j'}$ are known to some corrupt party, i.e., both $C_0 \cap T_j$ and $C_0 \cap T_{j'}$ are non-empty, the execution of $\Pi_{j,j'}$ is identical in both the simulation and a real execution. Furthermore, by statement (a) in Lemma 5, in each such $\Pi_{j,j'}$, all honest parties in $\mathcal{L}_7$ output valid CNF shares of $\gamma_j\lambda_{j'}$. The view of the adversary in the protocol and in the simulation differ only in that the view in each $\Pi_{j,j'}$ such that $T_j \cap C_0$ or $T_{j'} \cap C_0$ is empty. However, by statement (b) in Lemma 5, the view of the adversary is identically distributed irrespective of the value of γ_j and $\lambda_{j'}$. That these properties hold even when $\Pi_{j,j'}$ is executed in parallel for each j, j' can be argued using a hybrid argument along the lines of our previous proofs. This concludes the proof. $\square$

Executing this in parallel realizes a parallel multiplication functionality.

4.4 Realizing MPC from Layered Multiplication and Addition

In this section, we construct a secure (n, t, d)-layered protocol for computing any given function f by evaluating an layered arithmetic circuit computing the function. Suppose each party $\mathsf{P}_i^0, i \in [n]$ in the input layer has $x_i \in \mathbb{F}$ as input, and each party in the output layer (specified later) wants to compute $f(x_1, \ldots, x_n)$. The secure computation of f proceeds in three phases: an input sharing phase, a circuit evaluation phase and an output reconstruction phase.

In the input sharing phase, each input client verifiably CNF secret shares their input. In the circuit evaluation phase, the layered protocol traverses the layered circuit that evaluates f and evaluates every gate in the circuit. Evaluating a gate amounts to securely computing a CNF secret sharing of the value on the output wire of each gate using the CNF secret sharing of the values on its input wires. Finally, in the output reconstruction phase, the secret sharing of the value on the output wire is revealed to the output clients.

We elaborate on the circuit emulation phase below. Let C be a layered arithmetic circuit (Definition 9) over a field $\mathbb{F}$ with D layers that computes f. At the end of the input phase, the values on the input wires of all gates in layer one of C are simultaneously made available on the same layer of the protocol graph. In the circuit evaluation phase, the protocol keeps the invariant that, if a layer $i \in [D]$ of C is processed, then the values on all the output wires from layer i of C are simultaneously available of a specific layer of the protocol graph. The protocol can then process all gates in layer $i + 1$ of C preserving the invariant.

Recall that every gate in C is either a multiplication-by-constant gate, an addition gate or a multiplication gate. Given a CNF secret sharing of the value on the input wire(s) of a multiplication-by-constant gate or an addition gate, a secret sharing of the value on the output wire can be computed by locally processing the share. That is, the value on the output wire of the gate is available on the same layer (of the protocol graph) on which the values on the input wires have been secret shared. However, for a multiplication gate, computing a CNF secret sharing of the product of the values on the input wires using a t-secure protocol for multiplication consumes 7 layers. This poses a challenge when ensuring the invariant that the values on the output wires of all gates in a layer (of C) are made available on the same layer of the protocol graph. We get around this obstacle as follows: for a multiplication-by-constant or an addition gate G, after locally computing the secret sharing of the value on the output wire, we further compute a multiplication gate with value on the output wire of G as one input and the other input value being fixed to one (identity). This is achieved by using a trivial secret sharing of one as the other input and executing the layered protocol for multiplication which consumes $d = 7$ layers. Hence, we ensure the invariant we require. The protocol is formally described in Figure 4.7.

Protocol 4.7 (A layered protocol computing any n-ary function f)

PUBLIC PARAMETERS: A layered arithmetic circuit C over $\mathbb{F}$ with D levels that computes f. Output layer is $d = 7D + 6$.
SECRET INPUTS: Each input client $\mathsf{P}_i^0, i \in [n]$ has input x_i.
OUTPUTS: Each output client receives $f(x_1, \ldots, x_n)$.
SUBROUTINES: Protocols Π_{VSS}, and Π_{mult} realizing functionalities f_{VSS}, and f_{mult}, respectively.

Input sharing phase:

1. For each $i \in [n]$, execute in parallel Π_{VSS} with P_i^0 as dealer with input x_i and $\mathcal{L}_5$ as shareholders.

Circuit evaluation phase:

Invariant: For $1 \leq i \leq D$, parties in $\mathcal{L}_{5+7(i-1)}$ hold verifiable secret shares of the values on the input wires of any gate in layer i of C. The parties in $\mathcal{L}_{5+7(i-1)}$ process any gate G_j in layer i of C as follows.

(i). G_i is a multiplication-by-constant or addition gate:

1. Parties in $\mathcal{L}_{5+7(i-1)}$ locally compute the CNF secret shares of the value on the output wire of G_j using CNF secret shares of the value(s) of the input wires of G_j.
2. Securely compute the product of the value on the output wire of G_j and one. This is achieved as follows: Execute Π_{mult} with $\mathcal{L}_{5+7(i-1)}$ as input layer and $\mathcal{L}_{5+7i}$ as output layer. Parties in $\mathcal{L}_{5+7(i-1)}$ use the CNF secret sharing of the value on the output wire of G_j and a trivial CNF secret sharing of 1 as inputs to the computation.

(iii). G_i is a multiplication gate:

1. Execute Π_{mult} with $\mathcal{L}_{5+7(i-1)}$ as input layer as $\mathcal{L}_{5+7i}$ as output layer. Here, each party in $\mathcal{L}_{5+7(i-1)}$ uses their respective CNF shares of the input wires as inputs. Each party in $\mathcal{L}_{5+7(i-1)}$ receives CNF shares of the product of these values at the end of Π_{mult}.

Output reconstruction phase:

1. The parties in $\mathcal{L}_{5+7D}$ reveal the CNF shares of value on the output wire of the output gate to every output client in the next layer.
2. Each output client in $\mathcal{L}_{6+7D}$ recovers the value from the CNF shares.

Theorem 10. *Let f be an n-party functionality computed by a layered arithmetic circuit C over a finite field $\mathbb{F}$ and gates partitioned into layers $L_1, \ldots, L_D$. Then, for any $t < n/3$, the protocol in Figure 4.7 is an $(n, t, 6 + 7D)$-layered MPC protocol for f.*

Proof. We argue the security of the protocol in $(f^n_{\mathsf{VSS}}, f^n_{\mathsf{mult}})$-hybrid model and appeal to sequential composition to prove security of the entire protocol. This is possible since the protocol essentially uses execution of (parallel) Π_{VSS} from $\mathcal{L}_0$ to $\mathcal{L}_5$, and then only parallel invocations of Π_{mult} from

$\mathcal{L}_{5+7(i-1)}$ to $\mathcal{L}_{5+7(i)}$ for each $1 \leq i \leq D$. Thus, the protocol amounts to sequential invocations of secure protocols implementing parallel VSS, addition and multiplication functionalities.

Consider the protocol in $(f^n_{\mathsf{VSS}}, f^n_{\mathsf{mult}})$-hybrid model obtained from the protocol in Figure 4.7 by replacing executions of VSS and multiplication with f^n_{VSS} and f^n_{mult}, respectively. First, all inputs are CNF secret shared to $\mathcal{L}_5$. The protocol ensures that the value on the output wire of every gate is freshly and randomly secret shared before being consumed by gates in the next layer of the circuit. Thus, for each $1 \leq i \leq D$, parties in layer $5+7(i-1)$ hold fresh CNF shares of the values on the input wires of gates $G \in L_i$. Proceeding in this manner, it can be seen that the parties in layer $5+7D$ hold the value on the output wire of the circuit. The parties in $\mathcal{L}_{6+7D}$ recover this value by reconstructing the shares provided by the parties in the previous layer.

5 Efficient Layered MPC

In this section, we present an *efficient* implementation of perfectly t-secure layered MPC when $t < n/3$. To achieve this, we first build verifiable Shamir secret sharing. As in our previous implementation of MPC, the only non-trivial step in developing a protocol for general MPC after implementing VSS is that of achieving perfectly secure multiplication of two values that are secret shared. We build the multiplication protocol by porting a multiplication protocol of [CDN15, CDM00] from the standard setting to the layered setting. For want of space, we present the formal constructions and proofs of their security in the appendix. The security of the protocols is argued along the lines of our previous constructions, albeit, with slightly more complex proofs.

5.1 Verifiable Shamir Secret Sharing

In this section, we implement verifiable Shamir secret sharing in the layered setting with perfect t-security for $t < n/3$. This is realized by porting a protocol from the standard setting to the layered setting. We mostly face the same set of challenges that we encountered while implementing future multicast in the previous section. Recall that (t, n)-Shamir secret sharing of a secret s in a field $\mathbb{F}$ involves sampling a random polynomial $q(x)$ of degree at most t under the constraint $q(0) = s$ and setting the ith share to be $q(i)$. We consider an equivalent functionality $f_{\mathsf{ShamirVSS}}$ that allows a dealer to distribute the evaluation of a degree (at most) t polynomial on distinct non-zero points. A formal description of the parallel $f_{\mathsf{ShamirVSS}}$ functionality is given in Figure 5.1.

Figure 5.1 (Parallel $f_{\mathsf{ShamirVSS}}$ functionality for sharing polynomials of degree at most t.)

PUBLIC PARAMETERS: Dealers $\mathsf{S}_1, \ldots, \mathsf{S}_n \in \mathcal{L}_0$, shareholders $\mathsf{P}^6_1, \ldots, \mathsf{P}^6_n$. A finite field $\mathbb{F}$ of size more than n.

1. $f_{\mathsf{ShamirVSS}}$ receives coefficients $c_{i,l} \in \mathbb{F}$ for $0 \leq l \leq t$ from each $\mathsf{S}_i, i \in [n]$. $f_{\mathsf{ShamirVSS}}$ ignores any extra values sent by S_i and sets missing values as 0.
2. For each $i \in [n]$, define polynomial $q_i(x) = c_{i,t}x^t + \ldots + c_{i,1}x + c_{i,0}$, and send $q_i(j)$ to each $\mathsf{P}^6_j, j \in [n]$.

Implementing $f_{\mathsf{ShamirVSS}}$. The layered protocol realizing $f_{\mathsf{ShamirVSS}}$ is provided in Figure 5.2. We provide an outline and the intuition behind its construction.

The classic protocol for Shamir VSS in the standard setting proceeds as follows. Suppose the dealer wants to share a secret s from a field $\mathbb{F}$ such that $|\mathbb{F}| > n$ with t-security for $t < n/3$. The dealer samples a random bi-variate polynomial $S(x, y)$ of degree at most t in both the variables such that $S(0, 0) = s$, and transfers $f_i(x) = S(x, i)$ and $g_i(y) = S(i, y)$ to party P_i. If the polynomials were appropriately sampled, $f_i(j) = S(i, j) = g_j(i)$ for every i, j. Each pair of parties P_i, P_j check if $f_i(j) = g_j(i)$ and $f_j(i) = g_i(j)$; P_i raises a complaint by broadcasting $(i, j, f_i(j), g_i(j)$ if this check fails for P_j. The dealer addresses every valid complaint–a complaint of the form (i, j, u, v) such that $u \neq f_i(j)$ or $v \neq g_i(j)$–and broadcasts (f_i, g_i); otherwise, the dealer dismisses that complaint. This is followed by parties casting votes to accept or disqualify the dealer. P_i votes to accept the dealer if all the following conditions are met: dealer addressed one of every inconsistent mutual complaint–*i.e.*, a pair of complaints (i, j, u, v) and (j, i, u', v') such that $u \neq u'$ or $v \neq v'$; P_i itself did not issue a complaint; and for each broadcasted (f_j, g_j), $f_i(j) = g_j(i)$ and $g_i(j) = f_j(i)$. If the dealer receives less than $n - t$ votes, it is declared to be corrupt. Otherwise, each P_i updates (f_i, g_i) if it was broadcasted by the dealer and sets $f_i(0)$ as their share.

Using selective reveal in future messaging and checking equality using f_{add} as done in future multicast, we can port the above protocol into the layered setting. The protocol obtained in this manner is used as sub-protocol Π in our final construction in Figure 5.2. Interestingly, this construction by itself is not a layered protocol for verifiable secret sharing. However, Π guarantees the following: Let $H_1 \subseteq [n]$ such that P_i^1 is honest iff $i \in H_1$; parties in $\mathcal{L}_5$ hold a secret sharing of a value $\hat{s}_i$ such that, all such $\hat{s}_i$ (there are at least $n - t$ of them) define a valid secret sharing of a value $\hat{s}$. Further, if the dealer is honest, $\hat{s} = s$ and $\hat{s}_i$ is the same as the value that the dealer transferred to P_i^1. This is formally stated in Lemma 6.

Lemma 6. *The following properties hold for an execution of Π in the presence of a layered adversary $\mathcal{A}$:*

(a) Let $G \subseteq [n]$ such that P_i^1 is honest if and only if $i \in H_1$. There exist polynomials $\hat{g}(x)$ and $\hat{g}_i(x), i \in H_1$, each of degree at most t, such that $\hat{g}_i(0) = \hat{g}(i)$ and α_i^k output by each honest party P_k^5 coincides with $\hat{g}_i(k)$. Furthermore, if S is honest, $\hat{g}(x) = F(x, 0)$.

(b) If S is honest, for any $r, r' \in \mathbb{F}$,

$$\mathrm{ADVR}_{\Pi,\mathcal{A}}(r) \equiv \mathrm{ADVR}_{\Pi,\mathcal{A}}(r').$$

Protocol 5.2 (An $(n, t, 6)$-layered protocol for realizing $f_{\mathsf{ShamirVSS}}$.)

PUBLIC PARAMETERS: Dealer $\mathsf{S} \in \mathcal{L}_0$, shareholders $\mathsf{P}_1^6, \ldots, \mathsf{P}_n^6 \in \mathcal{L}_6$. A finite field $\mathbb{F}$, $|\mathbb{F}| > n$. A (t, n)-Shamir secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$.

SECRET INPUTS: Dealer holds a polynomial $q(x) = c_t x^t + \ldots + c_1 x + c_0$ of degree at most t over $\mathbb{F}$.

SUBROUTINES: Protocol Π_{FM} realizing f_{FM} functionality and Π_{add} realizing f_{add} functionality.

1. For each $0 \leq l \leq t$, concurrently execute protocol Π (described below) with c_i as input of S.
2. Each $\mathsf{P}_k^5, k \in [n]$ stores the output at the end of execution l of Π as $(\alpha_{l,i}^k)_{i\in[n]}$ for each $0 \leq l \leq t$. To each $\mathsf{P}_j^6, j \in [n]$, P_i^5 sends $\gamma_{i,j}^k = \sum_{l=0}^t \alpha_{l,i}^k j^l$ for each $i \in [n]$.
3. Each P_j^6 reconstructs $\gamma_{i,j} = \mathsf{Rec}(\gamma_{i,j}^1, \ldots, \gamma_{i,j}^n)$ for each $i \in [n]$. Finally, P_j^6 outputs $\gamma_j = \mathsf{Rec}(\gamma_{1,j}, \ldots, \gamma_{n,j})$.

SUB-PROTOCOL: Π with S as dealer with input s.

(i). Layer 0.

1. S samples a random bivariate polynomial $F(x, y)$ of degree at most t in both variables conditioned on $F(0, 0) = s$. Let $f_i(x) = F(x, i)$ and $g_i(y) = F(i, y)$ for each $i \in [n]$. S privately sends (f_i, g_i) to P_i^1.
2. For each $i, j \in [n]$, S samples $(F^1(i, j), \ldots, F^n(i, j)) \leftarrow \mathsf{Sh}(F(i, j))$. For each $k \in [n]$, S sends $F^k(i, j)$ to P_k^4 using future messaging (Π_{FM}).
3. For each $i \in [n]$, let $f_i(x) = \sum_{l=0}^t \alpha_{i,l} x^l$ and $g_i(x) = \sum_{l=0}^t \beta_{i,l} y^l$. S samples $(\alpha_{i,l}^1, \ldots, \alpha_{i,l}^n) \leftarrow \mathsf{Sh}(\alpha_{i,l})$, and $(\beta_{i,l}^1, \ldots, \beta_{i,l}^n) \leftarrow \mathsf{Sh}(\beta_{i,l})$ for each $0 \leq l \leq t$. For each $i, k \in [n]$ and $0 \leq l \leq t$, S sends $\alpha_{i,l}^k$ and $\beta_{i,l}^k$ to P_k^5 using future messaging.

(ii). Layer 1.

1. For each $i, j \in [n]$, execute Π_{add} to compute $f_i(j) - g_j(i)$ and $g_i(j) - f_j(i)$ and broadcast the results to $\mathcal{L}_3$.
2. P_i^1 samples $(f_i^1(j), \ldots, f_i^n(j)) \leftarrow \mathsf{Sh}(f_i(j))$, $(g_i^1(j), \ldots, g_i^n(j)) \leftarrow \mathsf{Sh}(g_i(j))$ for each $j \in [n]$, and sends both $f_i^k(j)$ and $g_i^k(j)$ to P_k^3 and to P_k^4 using future messaging.
3. P_i^1 samples $(g_i^1(0), \ldots, g_i^n(0)) \leftarrow \mathsf{Sh}(g_i(0))$ and sends $g_i^k(0)$ to P_k^5 using future messaging.

Figure 5.2 (Continued...)

(iii). Layer 3.
1. Each $\mathsf{P}^3_k, k \in [n]$ recovers
   (a) $F^k(i,j)$ as the output of future messaging initiated in step (i). 2. for each $i, j \in [n]$.
   (b) For each $j \in [n]$, P^3_k recovers $f^k_i(j)$ and $g^k_i(j)$ as the output of future messaging initiated in step (ii).2 by $\mathsf{P}^1_i, i \in [n]$.
2. Using the output of Π_{add}, P^3_k computes and broadcasts the set
$$S = \{\{i,j\} : f_i(j) - g_j(i) \neq 0 \text{ or } f_j(i) - g_i(j) \neq 0\},$$
and $F^k(i,j), F^k(j,i), f^k_i(j), g^k_i(j), f^k_i(j)$ and $g^k_i(j)$ for each $\{i,j\} \in S$.

(iv). Layer 4.
1. For each $\{i,j\} \in S$ and $l \in [n]$, $F^l(i,j)$ was broadcasted by $\mathsf{P}^3_l, l \in [n]$. Each $\mathsf{P}^4_k, k \in [n]$ recovers $F(i,j) = \mathsf{Rec}(F^1(i,j), \ldots, F^n(i,j))$. Similarly, P^4_k recovers $F(j,i)$, $f_i(j), g_i(j), f_j(i), g_j(i)$.
2. Each P^3_k recovers $\alpha^k_{i,l}$ and $\beta^k_{i,l}$ for each $0 \le l \le t$ as the output of future messaging initiated in step (i).3. with S as sender. P^3_k also recovers $f^k_i(j)$ and $g^k_i(j)$ for each $j \in [n]$ as the output of future messaging initiated in step (ii).2 with P^1_i as sender.
3. Define $B = \{i \in [n] : \exists j, \{i,j\} \in S \text{ and } (F(i,j) \neq g_i(j) \text{ or } F(j,i) \neq f_i(j))\}$. For each $i \in B$,
   (a) P^4_k broadcasts $\alpha^k_{i,l}$ and $\beta^k_{i,l}$ for each $0 \le l \le k$.
   (b) P^4_k broadcasts $f^k_j(i)$ and $g^k_j(i)$ for each $j \in [n]$.

(v). Layer 5.
1. For each $i \in B$ and $0 \le l \le t$, each P^5_k recovers $\alpha_{i,l} = \mathsf{Rec}(\alpha^1_{i,l}, \ldots, \alpha^n_{i,l})$ and $\beta_{i,l} = \mathsf{Rec}(\beta^1_{i,l}, \ldots, \beta^n_{i,l})$. Let $\hat{f}_i(x) = \sum_{l=1}^t \alpha_{i,l} x^l$ and $\hat{g}_i(y) = \sum_{l=1}^t \beta_{i,l} y^l$.
2. For each $i \in B$ and $j \in [n]$, each P^5_k recovers $f_j(i) = \mathsf{Rec}(f^1_j(i), \ldots, f^n_j(i))$ and $g_j(i) = \mathsf{Rec}(g^1_j(i), \ldots, g^n_j(i))$.
3. Define set $B' = \{j \in [n] : \exists i \in B, f_j(i) \neq \hat{g}_i(j) \text{ or } g_j(i) \neq \hat{f}_i(j)\}$.
   (a) If $|B \cup B'| \ge t$, dealer is disqualified. Each P^5_k outputs $(\alpha^k_i)_{i \in [n]}$, where $\alpha^k_i = 0$ for all $i \in [n]$.
   (b) Otherwise, for each $i \in [n] \setminus B$, P^5_k recovers $g^k_i(0)$ as the output of future messaging initiated in step (ii).3, and outputs $(\alpha^k_i)_{i \in [n]}$ such that $\alpha^k_i = g^k_i(0)$ if $i \in [n] \setminus B$ and $\hat{g}_i(0)$ otherwise.

Using Π as a subroutine, verifiable secret sharing is achieved as follows (in Figure 5.2. Let $q(x) = c_0 + c_1 x + \ldots c_t x^t$ be the polynomial that the dealer wants to secret share. For each $0 \le l \le t$, dealer S executes Π with c_i as its input. When $\mathsf{P}^5_i, i \in H_5$ are the set of honest parties in $\mathcal{L}_5$. By Lemma 6, for each $0 \le l \le t$, there exist polynomials $\hat{g}_l(x)$ and $\{\hat{g}_{l,i}(x)\}_{i \in H_1}$ of degree at most t such that, $\hat{g}_{l,i}(0) = \hat{g}_l(i)$, and for all $k \in H_5$ and $l \in H_1$, P^k_5 holds $\alpha^k_{l,i} = \hat{g}_{l,i}(k)$. Since $|H_5| \ge n - t$, each $\mathsf{P}^6_j, j \in [n]$ recovers
$$\gamma_{i,j} = \mathsf{Rec}(\alpha^1_{l,j}, \ldots, \alpha^n_{l,j}) = \sum_{l=0}^t \hat{g}_l(i) j^l, \forall i \in H_1$$
But then,
$$\gamma_j = \mathsf{Rec}(\gamma_{1,j}, \ldots, \gamma_{n,j}) = \sum_{l=0}^t \hat{g}_l(0) j^l.$$
Defining $\hat{q}(x) = \hat{g}_l(0) x^l$, we conclude that each P^6_j receives $\hat{q}(j)$ as required in verifiable Shamir secret sharing. When S is honest, by Lemma 6 (a), $\hat{g}_l(0) = c_l$ for each $0 \le l \le t$. Hence, $\hat{q}(x) = q(x)$.

We next argue that, when S is honest, the view of the adversary is identical irrespective of the value of $q(0)$. Assume that the guarantee in Lemma 6 (b) is preserved when Π is executed concurrently as in the protocol. Then, the view of the adversary till $\mathcal{L}_5$ are identically distributed in the protocol irrespective of the values of $(c_l)_{0 \le l \le t}$. Hence, the view of the adversary in the protocol only reveals $q(i)$ for $i \in C_6$, where $\mathsf{P}^6_i, i \in C_6$ are the set of corrupt parties in $\mathcal{L}_6$.

To formally argue the security of the protocol, we can build a simulator along the line of Theorem 7. Recall that, in the proof of Theorem 7, we used a hybrid argument to effectively extend the game based security guarantee of the weak multicast protocol $\Pi_{\mathsf{weak\text{-}FMcast}}$ as stated in Lemma 4 (b) to parallel invocations as used in the protocol in Figure 4.2. A similar argument can be used here.

In Protocol 5.2, the polynomial secret shared in $\mathcal{L}_6$ is exclusively determined by $\alpha^k_{l,i}$, for $i \in [n]$ and $0 \leq l \leq t$ stored by the honest parties P^5_k. In other words, the dealer is committed to polynomial $\hat{g}_l(x), 0 \leq l \leq t$ (as described in Lemma 6) when all the honest parties in $\mathcal{L}_5$ finish receiving messages from their predecessors. Furthermore, by Lemma 6, when S is honest, view of the layered adversary is identically distributed irrespective of input of S in each invocation of Π. This ensures that, when the protocol for verifiable secret sharing is executed in parallel, the polynomial being secret shared by a corrupt dealer cannot be correlated with that shared by an honest dealer. In the following theorem, we state this stronger result: when n verifiable secret sharing protocols are executed in parallel with $\mathsf{P}^0_i, i \in [n]$ as dealer and $\mathcal{L}_6$ as shareholders, we realize a parallel VSS functionality with t-security. The parallel VSS functionality $f_{\mathsf{ShamirVSS}}$ is formally described below.

Theorem 11. *The protocol in Figure 5.2 executed in parallel realizes $f_{\mathsf{ShamirVSS}}$ with perfect t-security for $t < n/3$ by consuming 6 layers, and by communicating $O(n^6)$ field elements over the point-to-point channels and $O(n^4)$ field elements over the broadcast channels for each secret.*

Employing the layered protocol for VSS, we proceed to port the protocol for secure computation in [CDN15] to the layered setting. An important functionality we use extensively for this transformation is *resharing*, which allows parties in $\mathcal{L}_a$ with (a valid) secret sharing of a secret s to "handover" the secret to parties in $\mathcal{L}_b$, for any $b > a$, by providing a fresh secret sharing of s. Using parallel invocation of VSS, realizing resharing is straight forward: secret shares of uniformly random secrets $c_l, 1 \leq l \leq t$ are made available on the input layer. Then, the secret s is reshared by distributing $f(i)$ to shareholder i in the output layer; here $f(x) = s + \sum_{l=1}^{t} c_l x^l$. Distributing secret shares of a uniformly random secret is achieved by having $t + 1$ parties in a previous layer secret share random secrets and the parties locally computing the shares of their sum (See functionality in Figure D.1 and its implementation in Figure D.2). The resharing functionality is formally defined in Figure D.3, and it is implemented as outlined above in Figure D.4.

5.2 Multiplication

The main challenge in realizing general MPC is securely implementing a multiplication protocol that computes a secret sharing of the product of two values using their shares. Following the outline of the MPC implementation in [CDN15], we first realize a simpler primitive, namely multiplication with helper, where the input clients hold secret sharing of a pair of values, and a special input client called the helper holds both values. This primitive allows the helper to verifiably secret share of the product of these values onto the output clients. The helper will be disqualified if the value secret shared is not the product.

Implementing multiplication with helper. We realize this functionality by porting a modified version of the implementation of the same in standard setting as presented in [CDN15]. The protocol in the standard setting works as follows: When α, β are the values to be multiplied, helper samples polynomials $f(x)$ and $g(x)$ of degree at most t conditioned on $f(0) = \alpha$ and $g(0) = \beta$. It then computes $h(x) = f(x)g(x)$; clearly, $h(0) = \alpha\beta$. It then verifiably secret shares $(\alpha_l)_{l\in[t]}, (\beta_l)_{l\in[t]}$ and $(\gamma_l)_{0\leq l\leq 2t}$, where

$$f(x) = \alpha + \sum_{l=1}^{t} \alpha_l x^l \qquad g(x) = \beta + \sum_{l=1}^{t} \beta_l x^l \qquad h(x) = \sum_{l=0}^{2t} \gamma_l x^l.$$

The parties now enter a verification phase in which $f(i), g(i)$ and $h(i)$ are revealed to P_i for each $i \in [n]$. P_i is to verify if $f(i)g(i) = h(i)$ and raise a complaint otherwise. For each complaint, $f(i), g(i)$ and $h(i)$ are publicly revealed; parties unanimously disqualify the helper if any of the complaint is valid. If all complaints turn out to be bogus, then $h(x)$ is verified to be $f(x)g(x)$ and $\gamma = \alpha\beta$. The parties now store the secret shares of γ as the shares of the product.

Our layered protocol follows the same logic with one notable difference. The helper in $\mathcal{L}_0$ secret shares the coefficients of $f(x), g(x)$ and $h(x)$ to $\mathcal{L}_6$ using the VSS protocol, with the exception of α and β. Recall that α and β are secret shared on $\mathcal{L}_0$; it is imperative to the correctness of the protocol that the secret shares of α and β provided to $\mathcal{L}_6$ are valid. But, this can be easily ensured by having

α and β in $\mathcal{L}_0$ reshared to $\mathcal{L}_6$. In the standard setting, this is realized by "transferring" the secret sharing of α and β to the helper; resharing ensures the same guarantees. By taking appropriate linear combinations of the coefficients of the polynomials, parties in $\mathcal{L}_6$ then reveal $f(i), g(i)$ and $h(i)$ to each $\mathsf{P}_i^7, i \in [n]$. Each P_i^7 raises a complaint if $f(i)g(i) \neq h(i)$ to $\mathcal{L}_8$. For each $i \in [n]$ with a complaint, parties in $\mathcal{L}_8$ selectively reveal $f(i), g(i)$ and $h(i)$ to all parties in $\mathcal{L}_9$. This is achieved by the trick we used in VSS as well as multicast and multiplication in the previous section. Finally, γ secret shared by the helper onto $\mathcal{L}_6$ is reshared to $\mathcal{L}_9$ and is used as the secret sharing of $\alpha\beta$ if the parties in $\mathcal{L}_8$ has not (unanimously) disqualified the helper.

When the helper is honest, throughout the protocol, the adversary only sees at most t shares of α, β, the evaluation of f, g and h on at most t points, and at most t shares of a sharing and resharing of γ. This ensures that the view of the adversary is identically distributed irrespective of the values of α and β. A corrupt helper is disqualified by the parties in $\mathcal{L}_8$ if and only if $h(x) \neq f(x)g(x)$. As we observed while analyzing the protocol for VSS, the sender commits to these coefficients by $\mathcal{L}_5$ as party of VSS protocol. Hence, when this protocol is executed in parallel, a corrupt helper is unable to correlate the event of their getting disqualified with the secret sharing of the product achieved in another parallel execution with an honest helper. Thus, the protocol remains secure under parallel composition. The protocol is formally described in Figure D.8.

Theorem 12. *The protocol in Figure D.8 realizes multiplication with helper functionality in Figure D.7 with perfect t-security for $t < n/3$.*

Multiplication. We proceed to the main primitive required to implement MPC–secure processing of the multiplication gate. Suppose two values α, β are Shamir secret shared using polynomials $f(x)$ and $g(x)$. Since $f(x)g(x)$ is a polynomial of degree at most $2t$, given $f(i)g(i)$ for at least $2t+1$ distinct $i \in [n]$, there exists a linear transformation that computes $f(0)g(0) = \alpha\beta$. For each $i \in [n]$, suppose we execute the multiplication with helper protocol from the previous section to verifiably secret shares the product $f(i)g(i)$ with the help of the party holding $f(i)$ and $g(i)$. The protocol guarantees that the secret sharing of the product is accepted (and the helper is not disqualified) whenever the helper adheres to the protocol; whereas, if the helper secret shares a value other than the product then the helper is disqualified. Since at least $n - t$ parties are corrupt, we obtain the correct secret sharing of $f(i)g(i)$ for $n - t \geq 2t + 1$ distinct values of i, which can be locally transformed using the aforementioned linear transformation to obtain a secret sharing of $\alpha\beta$.

The above proposal has a clear flaw: to multiply $f(i)$ and $g(i)$ held by a helper, both these values need to be secret shares in the same layer. Hence, we need each $f(i)$ and $g(i)$ to be further secret shared onto the input layer. But, this is exactly the 'data structure' guaranteed by reinforced secret sharing in Definition 10. Hence, we define the multiplication functionality to take *reinforced secret shares* of two values as input; to promote sequential processing of multiplication we also ensure that the output of the functionality is a reinforced secret sharing of the product of the input values. The multiplication functionality is formally described below:

Figure 5.3 (Multiplication functionality)

Public parameters: Output layer $\mathcal{L}_{10}$.
Input: Reinforced secret sharing of two values a, b;
i.e., $[[a, f(x), (f_i(x))_{i\in[n]})]]_0$ and $[[b, g(x), (g_i(x))_{i\in[n]})]]_0$.

1. Input clients reveal shares $\langle a, f(x)\rangle_0$ and $\langle b, g(x)\rangle_0$ to the functionality who reconstructs a, b.
2. Functionality distributes a reinforced secret sharing of ab onto the output clients. That is, it distributes $\langle ab, h(x)\rangle_{10}$, where $h(x)$ is a random polynomial of degree at most t such that $h(0) = ab$; for each $i \in [n]$, it distributes $\langle h(i), h_i(x)\rangle_b$, where $h_i(x)$ is a random polynomial of degree at most t such that $h_i(0) = h(i)$.

Implementing multiplication. The protocol takes valid reinforced secret sharing of two values as input. It then proceeds as outlined above by appropriately executing the multiplication with helper protocols and then linearly combining the evaluations on the product polynomial to obtain a *Shamir secret sharing* of the product of the inputs. It remains to convert the Shamir secret sharing to a reinforced secret sharing of the product; this is realized by executing a protocol for reinforced resharing implemented in Figure D.6. The protocol for secure multiplication is formally presented in Figure 5.4.

Figure 5.4 (A t-secure protocol for multiplication)

PUBLIC PARAMETERS: Output layer $\mathcal{L}_d$.
INPUT: $[[\alpha, f(x), (f_i(x))_{i\in[n]})]]_0$ and $[[\beta, g(x), (g_i(x))_{i\in[n]})]]_0$.
SUBROUTINES: Π_{multh} and Π_{rreshare} implementing multiplication with helper and reinforced resharing, respectively.
SETUP: Random secret shares in appropriate layers as required to execute multiplication with helper and reinforced resharing.

(i). Product computation

1. For each $i \in [n]$, execute Π_{multh} for multiplication with P_i^0 as helper to share $f(i)g(i)$ onto $\mathcal{L}_9$. The inputs to the protocol are $\langle f(i), f_i(x)\rangle_0$ and $\langle f(i), f_i(x)\rangle_0$. Additionally, the helper holds $f(i)$ and $g(i)$, the shares of P_i^0 in $\langle \alpha, f(x)\rangle_0$ and $\langle \beta, g(x)\rangle_0$, respectively.
2. Let $G \subseteq [n]$ such that, for each $i \in G$, at the end Π_{multh} with P_i^0 as helper, the parties stored $\langle f(i)g(i), h_i(x)\rangle_9$ as a valid secret sharing of $f(i)g(i)$ (instead of disqualifying the helper). There exist scalars $(r_i)_{i\in G}$ such that $\sum_{i\in G} r_i f(i)g(i) = f(0)g(0) = \alpha\beta$. Define

$$\langle \alpha\beta, h(x)\rangle_9 = \langle \sum_{i\in G} r_i f(i)g(i), \sum_{i\in G} r_i h_i(x)\rangle_9 = \sum_{i\in G} r_i \langle f(i)g(i), h_i(x)\rangle_9$$

(ii). Reinforced resharing of the product

1. Execute the t-secure protocol for reinforced resharing with $\langle \alpha\beta, h(x)\rangle_9$ as input from $\mathcal{L}_9$ and $\mathcal{L}_{10}$ as the output layer, and random secret shares distributed onto $\mathcal{L}_9$ as setup. That is, execute $[[\alpha\beta, h'(x), ((h_i')(x))_{i\in[n]}]]_{10} \leftarrow \langle \alpha\beta, h(x)\rangle_9$.
2. The parties in $\mathcal{L}_4$ store their respective shares of $[[\alpha\beta, h'(x), ((h_i')(x))_{i\in[n]}]]_{10}$.

The protocol inherits t-security from t-security of protocols implementing (parallel) multiplication with helper and reinforced resharing since the protocol essentially uses these protocols in parallel. Indeed, the protocol remains secure under parallel composition because both the subroutines remain secure under parallel composition.

Theorem 13. *The protocol in Figure 5.4 realizes multiplication functionality in Figure 5.3 with perfect t-security for $t < n/3$.*

5.3 MPC

In this section, we construct an efficient t-secure protocol for securely computing any given function f by evaluating a layered arithmetic circuit C computing the function. Suppose each party $\mathsf{P}_i^0, i \in [n]$ in the input layer has $z_i \in \mathbb{F}$ as input, and each party in the output layer (specified later) wants to compute $f(z_1, \ldots, z_n)$. Similar to our CNF secret sharing based construction, the secure computation of f proceeds in three phases: an input sharing phase, a circuit evaluation phase and an output reconstruction phase.

In the input sharing phase, each input client secret shares their input using reinforced secret sharing. In the circuit evaluation phase, the protocol keeps the invariant that, if a layer i of C is processed, then the values on all the output wires outgoing from layer i of C are simultaneously available of a specific layer of the protocol graph. Given a reinforced secret sharing of the value on the input wire(s) of a multiplication-by-constant gate or an addition gate, a secret sharing of the value on the output wire can be computed by locally processing the share. However, for a multiplication gate, computing a Shamir secret sharing of the product of the values on the input wires using a t-secure protocol for multiplication consumes 10 layers. Hence, we once again face the challenge of ensuring the invariant that the values on the output wires of all gates in a layer (of C) are made available on the same layer of the protocol graph. We get around this obstacle the same way we did in our previous construction: for a multiplication-by-constant or an addition gate G, after locally computing the reinforced secret sharing of the value on the output wire, we further compute a multiplication gate with value on the output wire of G as one output and the other value being fixed to one. This is achieved by taking a trivial secret sharing of one as the other input and executing the t-secure protocol for multiplication which consumes 10 layers. In this manner, we preserve the invariant we require.

In the protocol for secure computation of f, we will use the protocol for multiplication presented in Figure 5.4 and the protocol for reinforced resharing presented in Figure D.6. Both these subroutines use several random secret sharing and random secret sharing with various owners as setup. We generate the setup needed to execute these protocols using the random secret sharing protocol presented in Figure 5.4. We will argue that the protocol remains secure when the setup is computed in parallel. The protocol is formally described in Figure 5.5.

Theorem 14. *Let f be an n-party functionality computed by a layered arithmetic circuit C over a finite field $\mathbb{F}$, with D levels and M gates. Then, for any $t < n/3$, there is an $(n, t, 8 + 10D)$-layered MPC protocol for f in which the communication consists of $M \cdot O(n^9)$ field elements over the point-to-point channels and $M \cdot O(n^7)$ field elements over the broadcast channels.*

Protocol 5.5 (Efficient layered protocol for computing a function f)

Public parameters: A layered arithmetic circuit C over $\mathbb{F}$ with D levels that computes the desired function f. Output layer $10D + 7$.

Secret inputs: Each input client $\mathsf{P}_i^0, i \in [n]$ has input z_i.

Subroutines: protocols implementing VSS (Figure 5.2), random secret sharing (Figure D.2), reinforced resharing (Figure D.6) and multiplication (Figure 5.4).

Input sharing stage:

1. For each $i \in [n]$, execute verifiable secret sharing with P_i^0 as dealer with input z_i and $\mathcal{L}_7$ as shareholders to distribute $\langle z_i, p_i(x)\rangle_6$.
2. For each $i \in [n]$, execute reinforced resharing protocol from $\mathcal{L}_6$ to $\mathcal{L}_7$ to realize $[[z_i, f_i(x), (f_{i,j}(x))_{j\in[n]}]]_7 \leftarrow \langle z_i, p_i(x)\rangle_6$.

Circuit evaluation stage:

Invariant: For $1 \leq i \leq D$, parties in $\mathcal{L}_{7+10(i-1)}$ hold reinforced secret shares of the values on the input wires of any gate in layer i of C. The parties in $\mathcal{L}_{7+10(i-1)}$ process any gate G_j in layer i of C as follows:

(i). G_j is a multiplication-by-constant or addition gate:

1. Parties in $\mathcal{L}_{7+10(i-1)}$ locally compute the reinforced secret sharing $[[o_j, g_j(x), (g_{j,k})_{k\in[n]}]]_{7+10(i-1)}$ of the value on G_j's output wire.
2. Securely compute the product of value on the output wire of G_j with identity. This is achieved as follows: Execute the t-secure protocol for multiplication in with $\mathcal{L}_{7+10(i-1)}$ as input layer and $\mathcal{L}_{7+10i}$ as output layer. Parties in $\mathcal{L}_{7+10(i-1)}$ use $[[o_j, g_j(x), (g_{j,k})_{k\in[n]}]]_{7+10(i-1)}$ and a trivial reinforced secret share of 1 given by $[[1, 1(x), (1(x))_{i\in[n]}]]_{7+10(i-1)}$ as inputs to the computation. Here, $1(x)$ is the constant polynomial that evaluates to 1 on every point.

(iii). G_i is a multiplication gate:

1. Execute the protocol for multiplication with $\mathcal{L}_{7+10(i-1)}$ input layer and $\mathcal{L}_{7+10i}$ as output layer. Parties in $\mathcal{L}_{7+10(i-1)}$ use the reinforced secret shares of the values on the input wires as inputs for this computation.

Output reconstruction phase:

1. The parties in $\mathcal{L}_{7+10D}$ reveal the reinforced secret shares of the output wire of C to every output client in the next layer.
2. Each output client in $\mathcal{L}_{8+10D}$ recovers the value on the output wire from the reinforced secret shares.

Implementing setup:

1. Subroutines for reinforced resharing and for multiplication used setup at various levels. In order to obtain random secret sharing on $\mathcal{L}_a$, execute the t-secure protocol in Figure D.2. This protocol uses verifiable secret sharing to be invoked by parties $\mathsf{P}_i^{a-7}, i \in [t + 1]$. But, random secret sharing is invoked for the first time in $\mathcal{L}_7$.

6 Efficiency Improvement for Computational Security

We can use standard techniques to improve the efficiency of our constructions if the security guarantee is relaxed from perfect to computational while maintaining perfect correctness but achieving only static security. Verifiably secret sharing N distinct values using the scheme we presented in Figure 5.2 requires communicating $N \cdot O(n^6)$ field elements in total. We show a simple scheme that realizes secret sharing of random secrets with a preprocessing cost of $2^{O(n)}$ bits but uses no communication for subsequent sharing of unlimited random secrets.

Let f be a pseudorandom function that takes a λ bit long seed. When $\mathcal{T} = \{T \subset [n] : |T| = n - t\}$, the dealer uses future multicast to transfer a λ-bit seed s_T to parties $(\mathsf{P}^6_i)_{i \in T}$. This requires $2^{O(n)} \cdot \lambda$ communication in total. Next, to sample the secret share of the ith random secret r_i, each party computes $\delta_T = f_{s_T}(i)$ for each $T : i \in T$ and stores $(\delta_T)_{T:i\in T}$ as their CNF share. The dealer knows r_i since it has access to $(s_T)_{T\in\mathcal{T}}$, so it can broadcast a single value $s_i - r_i$ to $\mathcal{L}_6$ using $4n+1$ broadcasts, allowing the parties obtain shares of an arbitrary value s_i from their shares of r_i. In this protocol, a computationally bounded adversary only learns $s - r$, which is independent of s. Even when the dealer is corrupt, since r is correctly secret shared, the shares held by the parties remain valid at the end of this transformation. Asymptotically, this protocol making black-box use of a PRG uses $O(\ell)$ bits of communication per secret.

Theorem 15. *There exists a $(n, t, 5)$-layered protocol with* computational *t-security against a static malicious adversary and perfect t-correctness making black-box use of a PRG with seed length λ, that verifiable secret shares ℓ bits with $\lambda \cdot 2^{O(n)} + O(n\ell)$ bits of communication.*

We now obtain an asymptotically efficient layered protocol for general secure computation with computational t-security and perfect correctness for $t < n/3$. In our construction of Figure 5.5, the $O(n^3)$ invocations of VSS per gate dominate the communication cost. We tweak this protocol by realizing Shamir secret sharing by first distributing CNF shares using the VSS protocol of Theorem 15 and applying share conversion [CDI05] to obtain Shamir secret shares. The resulting protocol computes each gate using $O(n^5)$ bits of amortized communication.

Theorem 16. *Let f be an n-party functionality computed by a layered arithmetic circuit C over a finite field, with D levels and M gates. Then, for any $t < n/3$, there is an $(n, t, O(D))$-layered MPC protocol for f with* computational *t-security against a static malicious adversary and perfect correctness making black-box use PRG with seed length λ, and using $\lambda \cdot 2^{O(n)} + O(n^5 \cdot M)$ bits of communication.*

Given a Boolean circuit C computing a function f, a constant depth arithmetic circuit with $O(c)$ gates over a field of characteristic 2 that computes a garbled circuit of C can be constructed as in [DI05]. Using the protocol constructed in the above theorem to evaluate this circuit, we get the following result:

Corollary 3. *Let f be an n-party functionality computed by a Boolean circuit C with M gates. Then, for any $t < n/3$, there is an $(n, t, O(1))$-layered MPC protocol for f with* computational *t-security against a static malicious adversary and perfect correctness making black-box use PRG with seed length λ, and using $\lambda \cdot 2^{O(n)} + O(n^5 \cdot M)$ bits of communication.*

References

ADN06. J. F. Almansa, I. Damgård, and J. B. Nielsen. Simplified threshold RSA with adaptive and proactive security. In *EUROCRYPT 2006*, *LNCS* 4004, pages 593–611. Springer, Heidelberg, May / June 2006.

AHKP22. A. Acharya, C. Hazay, V. Kolesnikov, and M. Prabhakaran. SCALES - MPC with small clients and larger ephemeral servers. In *TCC 2022, Part II*, *LNCS* 13748, pages 502–531. Springer, Heidelberg, November 2022.

AL17. G. Asharov and Y. Lindell. A full proof of the bgw protocol for perfectly secure multiparty computation. *J. Cryptol.*, 30(1):58–151, jan 2017.

BELO15. J. Baron, K. El Defrawy, J. Lampkins, and R. Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In *ACNS 15*, *LNCS* 9092, pages 23–41. Springer, Heidelberg, June 2015.

BGG$^+$20. F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin. Can a public blockchain keep a secret? In *TCC 2020, Part I*, *LNCS* 12550, pages 260–290. Springer, Heidelberg, November 2020.

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

Can00. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.

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

CDD$^+$04. R. Canetti, I. Damgard, S. Dziembowski, Y. Ishai, and T. Malkin. Adaptive versus non-adaptive security of multi-party protocols. *Journal of Cryptology*, 17:153–207, 2004.

CDI05. R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, *Lecture Notes in Computer Science* 3378, pages 342–362. Springer, 2005.

CDM00. R. Cramer, I. Damgård, and U. M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, *Lecture Notes in Computer Science* 1807, pages 316–334. Springer, 2000.

CDN15. R. Cramer, I. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

CGG$^+$21. A. R. Choudhuri, A. Goel, M. Green, A. Jain, and G. Kaptchuk. Fluid MPC: Secure multiparty computation with dynamic participants. In *CRYPTO 2021, Part II*, *LNCS* 12826, pages 94–123, Virtual Event, August 2021. Springer, Heidelberg.

CH01. R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In *Advances in Cryptology—CRYPTO'94: 14th Annual International Cryptology Conference Santa Barbara, California, USA August 21–25, 1994 Proceedings*, pages 425–438. Springer, 2001.

CKLS02. C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97, 2002.

DEP21. I. Damgård, D. Escudero, and A. Polychroniadou. Phoenix: Secure computation in an unstable network with dropouts and comebacks. Cryptology ePrint Archive, Paper 2021/1376, 2021. https://eprint.iacr.org/2021/1376.

DI05. I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO 2005*, *LNCS* 3621, pages 378–394. Springer, Heidelberg, August 2005.

DJ97. Y. Desmedt and S. Jajodia. Redistributing secret shares to new access structures and its applications. Technical report, Citeseer, 1997.

DS83. D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

ELL20. K. Eldefrawy, T. Lepoint, and A. Leroux. Communication-efficient proactive secret sharing for dynamic groups with dishonest majorities. In *ACNS 20, Part I*, *LNCS* 12146, pages 3–23. Springer, Heidelberg, October 2020.

FL81. M. J. Fischer and N. A. Lynch. A lower bound for the time to assure interactive consistency. Technical report, GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION AND COMPUTER SCIENCE, 1981.

Gar94. J. A. Garay. Reaching (and maintaining) agreement in the presence of mobile faults. In *International Workshop on Distributed Algorithms*, pages 253–264. Springer, 1994.

GHK$^+$21. C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In *CRYPTO 2021, Part II*, *LNCS* 12826, pages 64–93, Virtual Event, August 2021. Springer, Heidelberg.

GHM$^+$17. Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

GIKR01. R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. The round complexity of verifiable secret sharing and secure multicast. In *33rd ACM STOC*, pages 580–589. ACM Press, July 2001.

GKM$^+$22. V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song. Storing and retrieving secrets on a blockchain. In *PKC 2022, Part I*, *LNCS* 13177, pages 252–282. Springer, Heidelberg, March 2022.

Gol09. O. Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

HIJ$^+$16. S. Halevi, Y. Ishai, A. Jain, E. Kushilevitz, and T. Rabin. Secure multiparty computation with general interaction patterns. In *ITCS 2016*, pages 157–168. ACM, January 2016.

HJKY95. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO'95*, *LNCS* 963, pages 339–352. Springer, Heidelberg, August 1995.

HM00. M. Hirt and U. M. Maurer. Player simulation and general adversary structures in perfect multi-party computation. *J. Cryptol.*, 13(1):31–60, 2000.

ISN89. M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1989.

KLR10. E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. *SIAM Journal on Computing*, 39(5):2090–2112, 2010.

Mau06. U. Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.

MZW$^+$19. S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song. CHURP: Dynamic-committee proactive secret sharing. In *ACM CCS 2019*, pages 2369–2386. ACM Press, November 2019.

OY91. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *10th ACM PODC*, pages 51–59. ACM, August 1991.

SLL10. D. Schultz, B. Liskov, and M. Liskov. Mpss: mobile proactive secret sharing. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):1–32, 2010.

WWW02. T. M. Wong, C. Wang, and J. M. Wing. Verifiable secret redistribution for archive systems. In *First International IEEE Security in Storage Workshop, 2002. Proceedings.*, pages 94–105. IEEE, 2002.

Supplementary Material

A Preliminaries

A.1 Secret Sharing

A central tool used for securely implementing basic primitives for layered MPC is secret sharing. A (t, n)-secure secret sharing scheme takes a secret s as input and computes n shares with the secrecy guarantee that any set of at most t shares reveal nothing about s while any set of $t+1$ correct shares can be used to efficiently reconstruct the secret. A robust secret sharing scheme additionally guarantees correct reconstruction from n shares, even when an arbitrary set of t out of the n shares have been tampered. Formally,

Definition 6 (Secret Sharing). *A secret sharing scheme with secret domain S and share domain U is a pair of algorithms $(\mathsf{Sh}, \mathsf{Rec})$ called the sharing algorithm and the reconstruction algorithm, respectively.*

- **Sharing.** Sh *takes as input a secret $s \in S$ and randomness ρ chosen uniformly from some domain R and computes $\mathsf{Sh}(s, \rho) = (s_1, \ldots, s_n)$, where $s_i \in U$ will be called the ith share of the secret s. If ρ is clear from the context we just write $\mathsf{Sh}(s) = (s_1, \ldots, s_n)$.*
- **Reconstruction.** Rec *takes the set of all shares and computes $\mathsf{Rec}(s_1, \ldots, s_n) = \hat{s}$.*

The (t, n)-secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ should satisfy the following properties:

Correctness. *For all $\rho \leftarrow R$, and any $s \in S$, we have $\mathsf{Rec}(\mathsf{Sh}(s, \rho)) = s$.*
t-security. *For any $s, s' \in S$, and $I \subset [n], |I| \leq t$,*

$$\left\{\{s_i\}_{i\in I} \middle| \rho \leftarrow R, \mathsf{Sh}(s, \rho) = \{s_i\}_{i\in[n]}\right\} \equiv \left\{\{s_i\}_{i\in I} \middle| \rho \leftarrow R, \mathsf{Sh}(s', \rho) = \{s_i\}_{i\in[n]}\right\}.$$

The scheme $(\mathsf{Sh}, \mathsf{Rec})$ is said to be robust against t corruptions if it additionally satisfies the following reconstruction property:

t-robustness. *For any $\rho \leftarrow R$, a secret $s \in S$, a set $I \subset [n]$ of size at most t, and shares $\hat{s}_i \in U$, for $i \in I$,*

$$\mathsf{Rec}(\{s_i\}_{i\in[n]\setminus I}, \{\hat{s}_i\}_{i\in I}) = s, \text{ where } \mathsf{Sh}(s, \rho) = (s_1, \ldots, s_n).$$

We next define two commonly used, robust secret sharing schemes that we extensively use in our constructions.

Definition 7 (Threshold CNF-Secret Sharing [ISN89]). *Let $t, n \in \mathbb{N}$ and s be an element in a ring $\mathbb{L}$. Let $\mathcal{T}$ be an enumeration of all $\binom{n}{n-t}$ subsets of size $n-t$ such that $\mathcal{T} = \{T \subseteq 2^{[n]} : |T| = n-t\} = \{T_1, \ldots, T_k\}$. A threshold CNF-secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ is defined as follows:*

- **Sharing.** *To share a secret s, first sample $\{r_T\}_{T\in\mathcal{T}}$ uniformly at random from $\mathbb{L}$ subject to $s = \sum_{T\in\mathcal{T}} r_T$ (i.e., obtain a k-out-of-k additive secret sharing of s). Let s_i consist of all shares r_T such that $T \in \mathcal{T}$ and $i \in T$. Then, $\mathsf{Sh}(s, \rho) = (s_1, \ldots, s_n)$.*
- **Robust Reconstruction.** *Let $\{s_i\}_{i\in[n]}$ be the set of input shares. For each $i \in [n]$ and $T \in \mathcal{T}$ such that $i \in T$, denote the additive share r_T from s_i as r_T^i. For each $T \in \mathcal{T}$, choose the unique value $\hat{r}_T$ such that $|\{i \in [n] : r_T^i = \hat{r}_T\}| \geq n - 2t$. Then, $\mathsf{Rec}(\{s_i\}_{i\in[n]}) = \sum_{T\in\mathcal{T}} \hat{r}_T$.*

Definition 8 (Shamir Secret Sharing).
For t, n such that $t < n/3$, a (t, n)-Shamir secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ is defined using a finite field $\mathbb{F}$ such that $|\mathbb{F}| > n$ and distinct non-zero field elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.

- **Sharing.** *Given a secret $s \in \mathbb{F}$, Sh samples $c_1, \ldots, c_t$ uniformly and independently from $\mathbb{F}$ and defines the polynomial $p(x) = s + \sum_{i=1}^{t} c_i x^i$. Then, $\mathsf{Sh}(s) = (s_1, \ldots, s_n)$, where $s_i = p(\alpha_i)$.*
- **Robust Reconstruction.** *Let $\{\hat{s}_i\}_{i\in[n]}$ be the set of received shares. Rec finds a polynomial $p(x)$ of degree at most t such that $|\{i : p(\alpha_i) \neq \hat{s}_i\}| \leq t$. Then, $\mathsf{Rec}(\{\hat{s}_i\}_{i\in[n]}) = p(0)$.*

A.2 Layered Circuits

Definition 9 (Layered Arithmetic Circuits). *Let $\mathbb{F}$ be a field and $C : \mathbb{F}^n \to \mathbb{F}^m$ be an arithmetic circuit over $\mathbb{F}$ of size $|C| = M$ with gates $g_1, \ldots, g_M$ of the following types.*

- **Addition:** *Given input wire values $x_1, \ldots, x_k \in \mathbb{F}$, the output of the gate is $z = \sum_{i\in[k]} x_i$.*
- **Multiplication-by-Constant:** *Given input wire value $x \in \mathbb{F}$ and a constant $c \in \mathbb{F}$ associated with the gate, output the value $z = c \cdot x$.*
- **Multiplication:** *Given input wire values x_1 and x_2 from $\mathbb{F}$, the output wire value is the product $z = x_1 \cdot x_2$.*

C is a layered arithmetic circuit if the gates can be partitioned into layers $L_0, \ldots, L_D$ such that for every layer $1 \leq i \leq D$ and every gate $g \in L_i$, all inputs to g are outputs of gates in L_{i-1}.

A.3 Layered Broadcast

The definition of layered MPC (Definition 1) assumes layer-to-layer broadcast. This turns out to be a necessary since deterministic broadcast is impossible in the layered setting even if only considering a fail-stop adversary. [Gar94] also explored the relation between an adversarial mobility and the ability to reach agreement in his work on Byzantine Agreement with Mobile Faults (MBA). They identified a mobile fault setting $\mathrm{MF}(\frac{t}{n-1}, \rho)$ parameterized by the adversarial threshold t of all n parties and "roaming speed" ρ. Indeed, the setting of $\mathrm{MF}(\cdot, 1)$ (full "roaming speed") is enough to render BA impossible. We now cast the result of [Gar94] in the setting of layered MPC.

Theorem 17. *Deterministic perfect broadcast for layered MPC is possible iff $t = 0$.*

The proof sketch below follows the technique of [FL81]. They use equivalence classes of executions to argue the lower bound of $t+1$ on the round complexity of unauthenticated broadcast. This was later extended to the authenticated setting in the seminal work of [DS83].

Proof Sketch. Assume the existence of a protocol Π for broadcast over a layered graph consuming d layers *i.e.* with output being obtained in layer $d+1$ and assume for simplicity that $n = 4$. Furthermore, we assume that a faulty party merely drops a subset of its outgoing messages and will otherwise follow the protocol.

The idea is to build a chain of "equivalent" executions where each execution has at most one faulty party in each layer and where (1) the first execution in the chain has 0 as the unique decision value (2) the last execution has 1 as the unique decision value (3) any two consecutive executions in the chain are indistinguishable to some honest party in the last layer.

Let the initial execution be the case where the everyone are honest and the dealer has input 0. We say that a party is "muted" if it has no outgoing messages. Switching the input of a muted party has no effect on protocol execution. The crux of the proof is to show a sequence of equivalent executions that mutes a party in $\mathcal{S}^0$ and flips its input to 1 and then un-mutes it without violating requirement (3).

B Basic Primitives

B.1 Details omitted from Section 3.1

We continue the discussion on parallel composition of f_{FM} from Section 3.1 in the remark below.

Remark 2. When our protocol implementing f_{FM} is composed in parallel, the resulting functionality is not the natural parallel composition of f_{FM} which takes the input from each sender to each receiver and delivers them. In fact, this functionality is impossible to realize even in the trivial case of messaging from one layer to the very next using the provided secure communication link. As an example, suppose communication from $\mathsf{S}_1 \in \mathcal{L}_0$ to $\mathsf{R}_1 \in \mathcal{L}_1$ and from $\mathsf{S}_2 \in \mathcal{L}_0$ to $\mathsf{R}_2 \in \mathcal{L}_1$ are composed in parallel. Now, a rushing adversary corrupting S_1 and R_2 can collect the message from S_2 to R_2 and set this as the message from S_1 to R_1. Interestingly, this limitation persists when parallely composing our protocol for realizing f_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$ (even for $d > 1$) with t-security for $t < n/3$. Interestingly, this limitation persists when parallely composing our protocol for realizing f_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$ (even for $d > 1$) with t-security for $t < n/3$. We demonstrate this for $d = 2$. In this case, the future messaging protocol proceeds as follows: The sender in $\mathcal{L}_0$ secret shares the message among the parties in $\mathcal{L}_1$ using a (t, n)-robust secret sharing scheme. Parties forward the shares they received from the sender to the receiver in $\mathcal{L}_2$ who reconstructs the secret from the shares. Since at least $n - t$ shares are forwarded unchanged by the honest parties in $\mathcal{L}_1$ the receiver correctly recovers the message; this follows from the secret sharing scheme being t-robust. However, a corrupt sender can distribute invalid shares to parties in $\mathcal{L}_1$. This allows the corrupt parties (colluding with the sender) to divulge shares that (together with shares forwarded by the honest parties) reconstruct to a value of their choosing, which they could decide on after collecting the views of corrupt parties in the next layer by rushing.

B.2 Proving Lemma 2

Lemma 2 (Layered protocol for f^n_{FM}). *Let* $(\mathsf{Sh}, \mathsf{Rec})$ *be a robust* (t, n) *secret-sharing scheme (Definition 6), the* (n, t, d)*-layered protocol in Figure 3.3 realizes the functionality* f^n_{FM} *in Figure 3.2 with perfect security for* $t < n/3$.

Proof. We prove the security of the protocol by presenting a simulator $\mathcal{S}$ which, given oracle access to f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$, simulates the view of the adversary interacting with the protocol in Figure 3.3.

The simulator $\mathcal{S}$ works in two phases: In the first phase, $\mathcal{S}$ emulates f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$. Suppose the adversary corrupts $\mathcal{I} = \{\mathsf{P}^\ell_k, k \in I\} \subset \mathcal{L}_\ell$. In the protocol, as leakage from f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$, the adversary expects $(s_{(i,j),k})_{k \in I}$ among the shares $(s_{(i,j),k})_{k \in [n]}$ sampled according to $\mathsf{Sh}(m_{(i,j)})$ for each sender S_i and receiver R_j. These messages are simulated (without knowing $m_{(i,j)}$) by sampling them according to $\mathsf{Sh}(m)$ for an arbitrary member of M. On receiving this leakage, the adversary sends all messages that each corrupt S_i intends to send over f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$. The simulator now has all the values needed to simulate the output of f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ to all corrupt parties.

In the second phase, simulator emulates f^n_{FM} from $\mathcal{L}_\ell$ to $\mathcal{L}_d$. For this, it first invokes f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$ to receive the message $m_{(i,j)}$ from every honest S_i to every corrupt R_j. The adversary expects $(s_{(i,j),k})_{k \in I}$ received during f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ in the first phase of the simulation, and $(s_{(i,j),k})_{k \notin I}$ it will receive in this phase to be jointly distributed according to $\mathsf{Sh}(m_{(i,j)})$. Recall that, $\mathcal{S}$ sampled $(s_{i,j),k})_{k \in I}$ according to $\mathsf{Sh}(m)$ in the previous phase. But, since $|I| \leq t$, the simulator can patch $(s_{(i,j),k})_{k \notin I}$ such that $(s_{(i,j),k})_{k \in [n]}$ is distributed according to $\mathsf{Sh}(m_{(i,j)})$; this follows from t-privacy of $(\mathsf{Sh}, \mathsf{Rec})$. After resampling $(s_{(i,j),k})_{k \notin I}$, the simulator can safely provide the leakage from f^n_{FM} from $\mathcal{L}_\ell$ to $\mathcal{L}_d$. At this point, the adversary reveals the set of all messages that each corrupt P^ℓ_k intends to send over f^n_{FM} from $\mathcal{L}_\ell$ to $\mathcal{L}_d$.

The simulator now holds all the messages that are needed to compute the messages from every corrupt sender to every receiver (corrupt or honest). Hence, it can extract the appropriate inputs that the corrupt parties need to input to f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$. It remains to argue that, for any honest S_i and honest R_j, the output of R_j in the protocol coincides with the input of S_i. This follows from t-robustness of $(\mathsf{Sh}, \mathsf{Rec})$ since each $s_{(i,j),k}$ sent by S_i via an honest P^ℓ_k (there are at least $n - t$ of them) correctly reaches R_j.

Formally, consider any layered adversary $\mathcal{A}$ that non-adaptively corrupts a set of parties $\mathcal{I} \subset \mathcal{L}_\ell$ such that $|\mathcal{I}| \leq t$ and interacts with parties executing Π^n_{FM}. We will show that, for any input $m_{(i,j)} \in M, i, j \in [n]$,

$$\mathrm{EXEC}_{f^n_{\mathsf{FM}}, \mathcal{S}, \mathcal{I}}(m_{(i,j)}, i, j \in [n]) \equiv \mathrm{EXEC}_{\Pi^n_{\mathsf{FM}}, \mathcal{A}, \mathcal{I}}(m_{(i,j)}, i, j \in [n]). \tag{1}$$

Here, we used the notation from [Can00] to denote the joint distribution of the honest parties' outputs and the output of the adversary. When input to Π is $(x_1, \ldots, x_n)$, the random variable $\mathrm{EXEC}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n)$ is the vector

$$\begin{aligned}\mathrm{EXEC}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n) = (&\mathrm{ADVR}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n), \\ &\mathrm{EXEC}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n)_1, \\ &\ldots \\ &\mathrm{EXEC}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n)_n\,)\end{aligned}$$

where $\mathrm{ADVR}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n)$ is the output of $\mathcal{A}$, and, $\mathrm{EXEC}_{\Pi, \mathcal{A}, \mathcal{P}}(x_1, \ldots, x_n)_i$ is the output of party P^d_i for each $i \in [n]$ at the end of the interaction between Π invoked with input $(x_1, \ldots, x_n)$ and $\mathcal{A}$.

Simulator $\mathcal{S}$ works as follows:

Figure B.1 (Simulator $\mathcal{S}$ for the protocol in Figure 3.3)

PUBLIC PARAMETERS: Senders $\mathsf{S}_1, \ldots, \mathsf{S}_n \in \mathcal{L}_0$, receivers $\mathsf{R}_1, \ldots, \mathsf{R}_n \in \mathcal{L}_d$ where $d > 0$.

ADDITIONAL INPUTS TO THE SIMULATOR: Set of corrupt parties $\mathcal{I}$, and oracle access to adversary $\mathcal{A}$.

1. Emulating f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$.
   (a) For each honest S_i, $\mathcal{S}$ samples $(s_{(i,j),1}, \ldots, s_{(i,j),n})$ according to $\mathsf{Sh}(m)$ for every $\mathsf{R}_j, j \in [n]$. Here, m is an arbitrary member of M. For each corrupt P^ℓ_k, $\mathcal{S}$ leaks $(s_{(i,1),k}, \ldots, s_{(i,n),k})$ to $\mathcal{A}$ as message that S_i intends to send to P^ℓ_k over f^n_{FM}.
   (b) For each corrupt S_i and each P^ℓ_k, $\mathcal{S}$ receives $(s_{(i,1),k}, \ldots, s_{(i,n),k})$ from $\mathcal{A}$ as the message that S_i intends to send to party P^ℓ_k over f^n_{FM}.
   (c) Finally, for each corrupt P^ℓ_k, $\mathcal{S}$ delivers $(s_{(i,1),k}, \ldots, s_{(i,n),k})$ to $\mathcal{A}$ as message from $\mathsf{S}_i, i \in [n]$ to P^ℓ_k over f^n_{FM}.
2. Emulating f^n_{FM} from $\mathcal{L}_\ell$ to $\mathcal{L}_d$.
   (a) For each honest S_i and corrupt R_j, $\mathcal{S}$ receives the message $m_{(i,j)}$ that S_i intends to send to R_j as leakage from f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$.
   (b) Let $I \subset [n]$ such that P^ℓ_k is corrupt if and only if $k \in I$. For each honest S_i and corrupt R_j, resample $(s_{(i,j),k})_{k \notin I}$ such that $(s_{(i,j),k})_{k \in [n]}$ is distributed according to $\mathsf{Sh}(m_{(i,j)})$. Such a "patching" is possible by t-privacy of $(\mathsf{Sh}, \mathsf{Rec})$.
   (c) For each honest P^ℓ_k and corrupt R_j, $\mathcal{S}$ delivers $(s_{(1,j),k}, \ldots, s_{(n,j),k})$ to $\mathcal{A}$ as message that P^ℓ_k intends to send to R_j over f^n_{FM}.
   (d) For each corrupt P^ℓ_k, $\mathcal{S}$ receives $(s_{(1,j),k}, \ldots, s_{(n,j),k})$ from $\mathcal{A}$ as message P^ℓ_k intends to send to each $\mathsf{R}_j, j \in [n]$ over f^n_{FM}. $\mathcal{S}$ updates the values of each $s_{(i,j),k}$ that was changed in this step.
   (e) For each $\mathsf{P}^\ell_k \in \mathcal{L}_\ell$ and corrupt R_j, $\mathcal{S}$ delivers $(s_{(1,j),k}, \ldots, s_{(n,j),k})$ as message from P^c_k to R_j over f^n_{FM}.
3. For each corrupt S_i, compute $m_{(i,j)} = \mathsf{Rec}(s_{(i,j),1}, \ldots, s_{(i,j),n})$ as the message for R_j. Simulator sends $m_{(i,j)}$ to f^n_{FM} (from $\mathcal{L}_0$ to $\mathcal{L}_d$) as the message S_i intends to send to R_j. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

We now argue that $\mathcal{S}$ satisfies Equation (1). We separately analyze the outputs of honest parties and view of the adversary in both scenarios.

In the protocol, an honest R_j outputs $\mathsf{Rec}(\hat{s}_{(i,j),1}, \ldots, \hat{s}_{(i,j),n})$ as the message from a sender S_i. In the simulation, output of R_j is the input of an honest S_i, but if S_i is corrupt, it is $\mathsf{Rec}(\hat{s}_{(i,j),1}, \ldots, \hat{s}_{(i,j),n})$ (as chosen by $\mathcal{S}$). In both scenarios, $\hat{s}_{(i,j),k}$ coincides with $s_{(i,j),k}$ sent by S_i to P^ℓ_k over f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ if P^ℓ_k is honest, and with $\hat{s}_{(i,j),k}$ sent by P^ℓ_k to R_j over f^n_{FM} from $\mathcal{L}_\ell$ to $\mathcal{L}_d$ if P^ℓ_k is corrupt. This directly implies that the output of R_j in both scenarios coincide when S_i is corrupt. When S_i is honest, $(s_{(i,j),1}, \ldots, s_{(i,j),n})$ are shares of its input and $\hat{s}_{(i,j),k}$ coincides with $s_{(i,j),k}$ for at least $n - t$ distinct values of k (since there are at most t corrupt parties in $\mathcal{L}_\ell$). Hence, by the robustness of the secret sharing scheme, the output of R_j coincides with the input of an honest S_i.

The view of $\mathcal{A}$ consists of the messages sent by corrupt parties in $\mathcal{L}_0$, messages sent and received by corrupt parties in $\mathcal{L}_\ell$ and the messages received by corrupt parties in $\mathcal{L}_d$. Note that, in step 3.(a). of the simulation, $(s_{(i,j),1}, \ldots, s_{(i,j),n})$ is sampled according to $\mathsf{Sh}(m_{(i,j)})$ for any honest S_i and corrupt R_j. Using this observation, and inspecting the protocol and its simulation, we conclude that the view accumulated by $\mathcal{A}$ in both scenarios differ *only* in the joint distribution of messages sent by each honest S_i to corrupt parties in $\mathcal{L}_\ell$ which they are expected to forward to honest receivers. Formally, the difference is in the generation of the joint distribution of $(s_{(i,j),k})$ for triples (i, j, k) such that $\mathsf{S}_i, \mathsf{R}_j$ are honest and P^ℓ_k is honest. In the protocol, honest S_i samples $(s_{(i,j),1}, \ldots, s_{(i,j)}, n)$ according to $\mathsf{Sh}(m_{(i,j)})$ for (honest) R_j and sends $s_{(i,j),k}$ to the corrupt party P^ℓ_k. Whereas, in the simulation, for each honest S_i and honest R_j, $\mathcal{S}$ samples $(s_{(i,j),1}, \ldots, s_{(i,j)}, n)$ according to $\mathsf{Sh}(m)$ (for some arbitrary m) and sends $s_{(i,j),k}$ to the corrupt party P^ℓ_k. But, there are at most t corrupt parties in $\mathcal{L}_\ell$. Since the secret sharing scheme is t-robust, any set of t shares is identically distributed irrespective of the secret. Hence, we conclude that the distribution of the view of $\mathcal{A}$ (jointly, with outputs of honest parties) coincide in both scenarios, concluding the proof. □

C Layered MPC based on CNF Secret Sharing

C.1 Proof of Lemma 4

Lemma 4. *For any $T \in \mathcal{T}$, the following properties hold for $\Pi_{\mathsf{weak\text{-}FMcast}}$ with U_T as intermediaries when executed in the presence of any adversary $\mathcal{A}$:*

(a) There exists $\hat{r}$ such that all honest receivers in R output $\hat{r}$ at the end of $\Pi_{\mathsf{weak\text{-}FMcast}}$. Furthermore, if S is honest, $\hat{r} = r$.
(b) If S, and all intermediaries and receivers are honest, for any $r, r' \in M$,

$$\mathrm{ADVR}_{\Pi,\mathcal{A}}(r) \equiv \mathrm{ADVR}_{\Pi,\mathcal{A}}(r').$$

Proof. By Lemma 3:(1), every honest party in $\mathcal{L}_3$ computes the same output at the end of Π_{add}. Hence, all ($n - t$ or more) honest parties broadcasts a complaint (or refrain from doing so) in unison. In the former case, all receivers get $\hat{r} = \mathsf{Rec}(\hat{r}_1, \dots, \hat{r}_n)$ from each honest party in $\mathcal{L}_4$ and output the unique value sent by at least $n - t$ parties. Thus, all honest receivers output $\hat{r}$. When S is honest, by Corollary 2:(1), in every Π_{FM} initiated in step (i).2 of $\Pi_{\mathsf{weak\text{-}FMcast}}$ from S to P^3_i, the output $\hat{r}_i$ coincides with sender's input r_i. Hence, each $\mathsf{P}^4_j, j \in [n]$ receives $\hat{r}_i = r_i$ from each honest P^3_i in the previous level. Consequently, $\hat{r} = \mathsf{Rec}(r_1, \dots, r_n) = r$ by t-robustness of the secret sharing scheme. Thus, (a) holds in this case.

Suppose all honest parties in $\mathcal{L}_3$ refrain from registering complaints. Each receiver in R gets $(\hat{r}^j_i, j \in [n])$ from every honest P^4_i and computes the output according to step (v).2. By Corollary 2:(1), in each Π_{FM} (initiated in step (ii).2) from an honest sender P^1_j to an honest receiver P^4_i, the output $\hat{r}^j_i$ coincides with the input r^j_i. Hence, by t-robustness of secret sharing, all honest receivers correctly recover $\mathsf{Rec}(\hat{r}^j_1, \dots, \hat{r}^j_n) = r^j$ for each honest P^1_j. Furthermore, if every honest party in layer 3 refrains from registering a complaint, then the output of Π_{add} computing $r^j - r^{j'}$ must have been zero for every pair $j, j' \in T$. By Lemma 3:(1), this specifically implies that r^j and $r^{j'}$ are identical for any pair of honest parties P^1_j and $\mathsf{P}^1_{j'}$. Thus, there exists r' such that $r^j = r'$ for all honest P^1_j. Finally, when S is honest, r' coincides with r, the input of S. Since the set of $n - 2t$ or more honest parties forms a majority of the intermediaries, all honest receivers output r'. This concludes the proof of (a).

Before proving (b), we informally argue that the sender's secret is not leaked when intermediaries and receivers are honest. Assume that the statements about security of Π_{FM} and Π_{add} in Corollary 2 and Lemma 3 hold even when they are composed as in $\Pi_{\mathsf{weak\text{-}FMcast}}$. Since $r^j = r$ for every intermediary P^1_j, $r^j - r^{j'} = 0$ for every pair j, j'. Consequently, every honest party in $\mathcal{L}_3$ refrain from broadcasting a complaint. Hence, the view of $\mathcal{A}$ interacting with $\Pi_{\mathsf{weak\text{-}FMcast}}$ with honest sender, intermediaries and receivers consists of the shares of r sent to the corrupt parties in $\mathcal{L}_3$ by S using Π_{FM} in step (i).2, and the shares of $r^j (= r)$ sent to corrupt parties in $\mathcal{L}_4$ by P^1_j using Π_{FM} initiated in step (ii).2. But, since the secret sharing scheme is t-secure, these shares do not reveal any information about r to the adversary. Thus, r remains private from the adversary.

We proceed to formally prove Lemma 4:(b) using a hybrid argument. Fix $r, r' \in M$ where $r \neq r'$. Let Θ (resp. Θ') be the protocol $\Pi_{\mathsf{weak\text{-}FMcast}}$ with $U_T = \{\mathsf{P}^1_i : i \in T\}$ as intermediaries and r (resp. r') as input of S as described in Figure 4.2. We progressively transform Θ to obtain Θ' and argue that the adversary's view is identically distributed across each of these transformations, proving (b).

Transformation $\Theta \to \Theta_1$. Suppose w.l.o.g. suppose P^3_i is honest if $i \in [k]$, where $k \geq n - t$. Consider a sequence of protocols $(\Theta_{0,i})_{0 \leq i \leq k}$, where $\Theta_{0,0}$ is identical to Θ. For $i \geq 1$, $\Theta_{0,i}$ is obtained by replacing $(r_1, \dots, r_n)$ in step (i).2 with $(r'_1, \dots, r'_i, r_{i+1}, \dots, r_n)$, where $(r'_1, \dots, r'_k, r_{k+1}, \dots, r_n)$ is distributed according to $\mathsf{Sh}(r')$. Note that $(r'_1, \dots, r'_k)$ can be sampled in this manner since the secret sharing scheme is t-secure and $k \geq n - t$. Finally, define $\Theta_1 = \Theta_{0,k}$. Notice that, in Θ_1, S sends a secret sharing of r' in step (i).2. The following claim suffices to prove that

$$\mathrm{ADVR}_{\Theta,\mathcal{A}}(\perp) \equiv \mathrm{ADVR}_{\Theta_1,\mathcal{A}}(\perp). \tag{2}$$

Here the argument is $\perp$ because neither protocols take any inputs.

Claim. For any $1 \leq \ell \leq k$,

$$\mathrm{ADVR}_{\Theta_{0,\ell-1},\mathcal{A}}(\bot) \equiv \mathrm{ADVR}_{\Theta_{0,\ell},\mathcal{A}}(\bot).$$

Proof. The only difference between $\Theta_{0,\ell-1}$ and $\Theta_{0,\ell}$ is that the sender's input in the instance ℓ of Π_{FM} from S to P^3_ℓ is r_ℓ in the former and r'_ℓ in the latter. Since the sender and all intermediaries are honest, as we previously observed, all honest parties in $\mathcal{L}_3$ refrain from broadcasting a complaint and, more importantly, refrain from broadcasting the output of Π_{FM} from S initiated in step (i).2. We will show that, if Equation (2) does not hold, then Corollary 2:(2) is not satisfied for Π_{FM} from S to P^3_ℓ; a contradiction.

Adversary $\mathcal{A}'$ with auxiliary input $\{r_i, i \in [n]\}, \{r'_i, i \in [k]\}$ non-adaptively corrupting the same set of parties that $\mathcal{A}$ corrupted in $\mathcal{L}_1$ and $\mathcal{L}_2$ interacts with an execution of Π_{FM} from honest S to honest P^3_i as follows:

- Let $\mathcal{I}_i \subset \mathcal{L}_i$, $0 \leq i \leq 5$ be the set of parties corrupted by $\mathcal{A}$. $\mathcal{A}'$ generates dummy corrupt parties $\mathcal{I}_i = \{\mathsf{P}^i_j : \mathsf{P}^i_j \in \mathcal{I}_i, i \in \{0,3,4,5\}\}$ and initialize their random tapes.
- $\mathcal{A}'$ hands over the control of corrupt parties $\mathcal{I}_i, i = 1,2$ and dummy corrupt parties $\mathcal{I}_i, i \in \{0,3,4,5\}$ to $\mathcal{A}$ which it internally invokes. $\mathcal{A}'$ also generates dummy honest parties $\mathcal{H}_i = \mathcal{L}_i \setminus \mathcal{I}_i$ for each $0 \leq i \leq 5$.
- Using the values of r_i, r'_i, $\mathcal{A}'$ emulates dummy honest parties $\mathcal{H}_i$ for each $0 \leq i \leq 5$ executing $\Theta_{0,\ell-1}$ and interacts with $\mathcal{A}$. However, it does not emulate the instance of Π_{FM} from S to P^3_ℓ initiated in step (i).2. of $\Theta_{0,\ell-1}$. Instead, $\mathcal{A}'$ interacts with the real execution of Π_{FM} in parallel, and forwards the messages from (and to) the honest parties in the real execution to (and from) the corrupt parties $\mathcal{I}_1 \cup \mathcal{I}_2$ who $\mathcal{A}'$ has handed over to $\mathcal{A}$. We stress that the output of real P^3_ℓ at the end of the execution Π_{FM} from S to P^3_ℓ is not needed by $\mathcal{A}'$ for this emulation. This is because, in $\Theta_{0,\ell-1}$ all honest parties, specifically P^3_ℓ, refrains from revealing the output of Π_{FM} from S (from step (i).2).
- At the end of $\Theta_{0,\ell-1}$, $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs.

In the instance of Π_{FM} from S to P^3_ℓ in step (i).2, if the input S is r_ℓ, the above interaction is identical to the interaction with parties executing $\Theta_{0,\ell-1}$ for adversary $\mathcal{A}$ corrupting $\mathcal{I}_i, 0 \leq i \leq 5$. This can be verified as follows: In the former, all instructions in $\Theta_{0,\ell-1}$ except the ones that are part of Π_{FM} from S to P^3_ℓ are carried out by dummy honest parties and corrupt parties controlled by $\mathcal{A}$. Whereas, in the latter, they carried out by the real honest parties and corrupt parties controlled by $\mathcal{A}$. But, $\mathcal{A}'$ emulates all the dummy parties honestly using freshly chosen randomness in the former case just as the honest parties will in the latter case. Thus,

$$\mathrm{ADVR}_{\Pi_{\mathsf{FM}},\mathcal{A}'}(r_\ell) \equiv \mathrm{ADVR}_{\Theta_{0,\ell-1},\mathcal{A}}(\bot).$$

As previously observed, $\Theta_{0,\ell-1}$ and $\Theta_{0,\ell}$ differ only in the input of the sender in Π_{FM} from S to P^3_ℓ. Hence, in the instance of Π_{FM} from S to P^3_ℓ in step (i).2, if the input S is r'_ℓ, the above interaction is identical to the interaction with parties executing $\Theta_{0,\ell}$ for adversary $\mathcal{A}$ corrupting $\mathcal{I}_i, 0 \leq i \leq 5$. Thus,

$$\mathrm{ADVR}_{\Pi_{\mathsf{FM}},\mathcal{A}'}(r'_\ell) \equiv \mathrm{ADVR}_{\Theta_{0,\ell},\mathcal{A}}(\bot).$$

But, since S and P^3_ℓ are honest, by Corollary 2,

$$\mathrm{ADVR}_{\Pi_{\mathsf{S}},\mathcal{A}'}(m'_j) \equiv \mathrm{ADVR}_{\Pi_j,\mathcal{A}}(\bot).$$

This concludes the proof of the claim. □

Transformation $\Theta_1 \to \Theta_2$. We construct Θ_2 from Θ_1 by setting $r^j = r'$ (instead of $r^j = r$) for each party P^1_j in step (ii).1. Intuitively, since all intermediaries are honest, parties in $\mathcal{L}_3$ only learn $r^j - r^{j'}$, and hence the view of $\mathcal{A}$ will be identical in both cases. To formally prove this, we again consider a sequence of hybrids which progressively transforms Θ_1 to Θ_2. In every new hybrid, we change the inputs used by a new pair of honest parties $\mathsf{P}^1_j, \mathsf{P}^1_{j'}$ $(j < j')$ to Π_{add} for computing $r^j - r^{j'}$ (in step (i).1) from $r^j = r, -r^j = -r$, respectively, to $r^j = r', -r^j = -r'$. Clearly, the final protocol in this sequence is exactly Θ_2. The difference between any pair of consecutive hybrids is the replacement of the inputs of a specific pair of honest parties $\mathsf{P}^1_j, \mathsf{P}^1_{j'}$ $(j < j')$ to Π_{add} from $r^j = r, -r^j = -r$, respectively, to $r^j = r', -r^j = -r'$. We argue that the view of $\mathcal{A}$ remains identically distributed across any pair of consecutive hybrids by appealing to Lemma 3 using the same line of arguments as in the above claim. We thus obtain,

$$\mathrm{ADVR}_{\Theta_1,\mathcal{A}}(\perp) \equiv \mathrm{ADVR}_{\Theta_2,\mathcal{A}}(\perp). \tag{3}$$

Transformation $\Theta_2 \rightarrow \Theta_3$. To obtain Θ_3 from Θ_2, for each $i \in T$, we replace $(r_1^i, \ldots, r_n^i)$ in step (ii).2 with $(s_1^i, \ldots, s_n^i)$, where $(s_1^i, \ldots, s_n^i)$ is distributed according to $\mathsf{Sh}(r')$ and $r_j^i = s_j^i$ for each j such that $\mathsf{P}_j^4 \in \mathcal{L}_4$ is corrupt. This is possible since the secret sharing scheme is t-secure and there are at most t corrupt parties in $\mathcal{L}_4$. Arguing that the adversary's view remains the same across this transformation is similar to arguing the same for transformation from Θ_0 to Θ_1. We thus obtain,

$$\mathrm{ADVR}_{\Theta_2,\mathcal{A}}(\perp) \equiv \mathrm{ADVR}_{\Theta_3,\mathcal{A}}(\perp). \tag{4}$$

Θ_3 can be transformed to Θ' by replacing r with r' in step (i).1. Note that this does not make any difference to the step (ii), which has already been modified to replace r with r'. Thus, from Equations (2) to (4), we conclude that

$$\mathrm{ADVR}_{\Theta,\mathcal{A}}(\perp) \equiv \mathrm{ADVR}_{\Theta',\mathcal{A}}(\perp).$$

This proves (b) concluding the proof. □

C.2 Constructing $\mathcal{S}$ for Theorem 7

Theorem 7. *Protocol Π_{FMcast} in Figure 4.2 is a secure $(n, t, 5)$-layered protocol realizing f_{FMcast} with input client S and output clients in R.*

Proof. When S is corrupt, $\mathcal{S}$ emulates the honest parties according to the instructions in Π_{FMcast} and interact with the adversary. Observe that none of the honest parties have inputs since S is the only party with input. For each $T \in \mathcal{T}$, $\mathcal{S}$ extracts an input r_T of S in $\Pi_{\mathsf{weak\text{-}FMcast}}$ with intermediaries U_T as described below: We observed in the proof of statement (a) of Lemma 4 that, at the end of $\Pi_{\mathsf{weak\text{-}FMcast}}$, all receivers output $\hat{r}$ computed in step (iv).1 of the protocol if all the honest parties in $\mathcal{L}_3$ broadcasted a complaint, and otherwise–i.e., if no honest party in $\mathcal{L}_3$ broadcasted a complaint–all receivers output the unique value that was received by all honest intermediaries. Hence, if all emulated honest parties in $\mathcal{L}_3$ broadcasted a complaint, $\mathcal{S}$ chooses r_T to be $\hat{r}$ (computed by emulated honest parties), and otherwise it choose r_T as the unique value received by all emulated honest intermediaries. Finally, it sends $\sum_{T\in\mathcal{T}} r_T$ to f_{FMcast} as the input of S. Security follows from statement (a) in Lemma 4.

Next, suppose S is honest. In this case, $\mathcal{S}$ emulates all the honest parties and interacts with $\mathcal{A}$. For this, $\mathcal{S}$ sets the input of S to an arbitrary fixed message $m^* \in M$. Since the output of all honest receivers coincides with the input of the sender, it suffices to show that the output of $\mathcal{A}$ is identically distributed while interacting with Π_{FMcast} or with $\mathcal{S}$.

Let $T^* \in \mathcal{T}$ such that $U_{T^*} = \{\mathsf{P}_i^1 : i \in T^*\}$ are honest. The shares $(r_T, T \in \mathcal{T} \setminus \{T^*\})$ used in the simulation is identically distributed as $(r_T, T \in \mathcal{T} \setminus \{T^*\})$ used in Π_{FMcast}. This is because, in step 1 of Π_{FMcast}, $(r_T, T \in \mathcal{T})$ is chosen to be an additive secret sharing of the input of S. Hence, for each $T \in \mathcal{T} \setminus \{T^*\}$, execution of $\Pi_{\mathsf{weak\text{-}FMcast}}$ with U_T as intermediaries in the presence of $\mathcal{A}$ is identical in both the simulation and in Π_{FMcast}. However, in $\Pi_{\mathsf{weak\text{-}FMcast}}$ with U_{T^*} as intermediaries, the input of S is $m^* - \sum_{T\in\mathcal{T}\setminus\{T^*\}} r_T$ in the former and $m - \sum_{T\in\mathcal{T}\setminus\{T^*\}} r_T$ in the latter.

If all the receivers are honest, by statement (b) of Lemma 4, the view of the adversary in $\Pi_{\mathsf{weak\text{-}FMcast}}$ with U_{T^*} as intermediaries is identical in both the simulation and in Π_{FMcast}, proving security.

However, when there are corrupt parties in R, the view of $\mathcal{A}$ additionally contains the view of the corrupt receivers. The view of corrupt receivers in $\Pi_{\mathsf{weak\text{-}FMcast}}$ with U_{T^*} as intermediaries needs to be made consistent with $r_{T^*} = m - \sum_{T\in\mathcal{T}\setminus\{T^*\}} r_T$ even though $\mathcal{S}$ chose $r_{T^*} = m^* - \sum_{T\in\mathcal{T}\setminus\{T^*\}} r_T$ in the simulation. To ensure this, $\mathcal{S}$ tweaks the messages from the emulated honest parties in $\mathcal{L}_4$ to the corrupt receivers in R as follows: $\mathcal{S}$ receives the input m of S from f_{FMcast}. In $\Pi_{\mathsf{weak\text{-}FMcast}}$, for each intermediary $\mathsf{P}_i^1 \in U_{T^*}$, $\mathcal{S}$ samples $(\hat{r}_1^i, \ldots, \hat{r}_n^i)$ according to distribution $\mathsf{Sh}(m - \sum_{T\in\mathcal{T}\setminus\{T^*\}} r_T)$ conditioned on $\hat{r}_j^i = r_j^i$ for each j such that P_j^4 in $\mathcal{L}_4$ is corrupt. This is possible since the secret sharing scheme is t-secure and there are at most t corrupt parties in each layer. In step (iv).2 (note that parties choose this step since all intermediaries and S are honest as observed in the proof of Lemma 4), each emulated honest party sends $\hat{r}_j^i$ instead of r_j^i. This ensures that the view of corrupt receivers in $\Pi_{\mathsf{weak\text{-}FMcast}}$ with U_{T^*} as intermediaries is consistent with the input of S. This concludes the proof. □

C.3 Constructing $\mathcal{S}$ for Theorem 8

Theorem 8. *The protocol in Figure 4.3 executed in parallel realizes $f_{\mathsf{parallel\text{-}VSS}}$ with perfect t-security for $t < n/3$ by consuming 5 layers, and by communicating $\binom{n}{t}^3 \cdot O(n^2)$ field elements over the point-to-point channels and over the broadcast channels for each secret.*

Proof. We now formally describe the simulator. The parallel VSS protocol with P_i^0 as dealer consists of $|\mathcal{T}|$ parallel multicast protocols. For $T \in \mathcal{T}$, dealer multicasts r_T^i to receivers $\mathsf{P}_j^5, j \in T$. The simulator emulates all the honest parties in each of these multicast protocols and interacts with the adversary. For this, each honest dealer (the honest dealers are the only honest parties with input) are initialized with uniformly random inputs for every multicast in which they act as sender. For each corrupt dealer P_i^0, the simulator extracts $\hat{r}_T^i$ as the input in the multicast from P_i^0 to receivers $\mathsf{P}_j^5, j \in T$ for each $T \in \mathcal{T}$. The simulator then sets $(\hat{r}_T^i)_{T\in\mathcal{T}}$ as the input of corrupt P_i^0 to $f_{\mathsf{parallel\text{-}VSS}}$.

We now argue that the view of the adversary is identically distributed when interacting with the simulator and with honest parties executing the protocol for parallel VSS. Suppose $\mathsf{P}_j^5, j \in T^*$ are honest (such T^* exists since the adversary corrupts at most t parties in each layer) and let $\mathcal{T}^* = \mathcal{T} \setminus \{T^*\}$. For any honest dealer P_i^0 and any input to the dealer, the additive share $r_{T^*}^i$ of the input is sent only to honest receivers. Since the additive shares $(r_T^i)_{T\in\mathcal{T}^*}$ are uniformly distributed, the only difference between the real protocol and the simulation is that honest P_i^0 with input m uses $r_{T^*}^i = m - \sum_{T\in\mathcal{T}^*} r_T^i$ in the real execution, whereas it uses $r_{T^*}^i = m^* - \sum_{T\in\mathcal{T}^*} r_T^i$ in the simulation, where m^* is the arbitrary input that the simulator assigns to P_i^0. Thus, security amounts to showing that the view of the adversary is identically distributed in the multicast protocol with honest sender and receivers irrespective of sender's input, which follows from Theorem 7:(b). Note that, this can be shown to hold for several parallel executions of multicasts using a hybrid argument in which the inputs in the multicast protocols are progressively changed one at a time to move from one set of inputs to a different set of inputs to the senders in the parallel multicasts. Finally, it needs to be shown that the extracted inputs for the corrupt parties are independent of the shares chosen by the honest parties in VSS. This follows from the fact that the view of corrupt parties in $\mathcal{L}_0, \ldots, \mathcal{L}_4$ does not reveal the message that an honest sender multicasts to corrupt receivers and shown in the proof of Theorem 7.

C.4 Details omitted from Section 4.3

Protocol C.1 (A t-secure protocol for multiplication *in the standard setting*)

INPUTS: Party $P_i, i \in [n]$ has inputs (r_i, s_i), where $(r_i)_{i\in[n]}$ and $(s_i)_{i\in[n]}$ are CNF secret shares of r and s, respectively.
DEFINITIONS: Let $\mathcal{T} = \{T_1, \ldots, T_N\} = \{T \subset [n] : |T| = n - t\}$.
For each $i \in [n]$, $s_i = (\gamma_j : i \in T_j, j \in [N])$
and $r_i = (\lambda_j : i \in T_j, j \in [N])$, where $\sum_{j\in[N]} \gamma_j = r$ and $\sum_{j\in[N]} \lambda_j = s$.

1. Party $P_i, i \in [n]$ verifiably shares $\gamma_j \cdot \lambda_{j'}$ for each (j, j') such that P_i has both γ_j and $\lambda_{j'}$. Party P_i has γ_j (resp. λ_j) if $i \in T_j$.
2. For each $(j, j') \in [N] \times [N]$, let $P_i, i \in I$ be the set of parties that secret shared $\gamma_j \cdot \lambda_{j'}$. Fix $i^* \in I$; for all $i \in I, i \neq i^*$, publicly check if VSS of $\gamma_j \cdot \lambda_{j'}$ provided P_{i^*} and P_i are equal. This amounts to securely computing the differences between the values CNF shared by both parties. If true, take the VSS provided by P_{i^*} as the secret sharing of $\gamma_j \cdot \lambda_{j'}$. Otherwise, disclose γ_j and $\lambda_{j'}$ by having all parties reveal γ_j and $\lambda_{j'}$. The secret sharing of $\gamma_j \cdot \lambda_{j'}$ is trivially computed from $\gamma_j \cdot \lambda_{j'}$.
3. Each party computes its share of $r \cdot s$ as the sum of their respective shares of $\gamma_j \cdot \lambda_{j'}$ for all $j, j' \in [N] \times [N]$.

C.5 Proof of Lemma 5

Lemma 5. *For any $j, j' \in [N]$, the following properties hold for $\Pi_{j,j'}$ when executed in the presence of an adversary $\mathcal{A}$:*

(a) There exists $(\delta_k)_{k\in[N]}$ such that $\sum_{k=1}^{N} \delta_k = \lambda_j\gamma_{j'}$, and each honest party $\mathsf{P}_i^7, i \in [N]$ outputs $(\delta_k)_{k:i\in T_k}$ at the end of $\Pi_{j,j'}$.
(b) Suppose parties $\mathsf{P}_i^0, i \in H$ are honest, then for any a, b, a', b',

$$\mathrm{ADVR}_{\Pi_{j,j'},\mathcal{A}}(\gamma_j = a, \lambda_{j'} = b) \equiv \mathrm{ADVR}_{\Pi_{j,j'},\mathcal{A}}(\gamma_j = a', \lambda_{j'} = b').$$

Proof. By the linearity of CNF secret sharing, and its t-robustness, all honest parties in $\mathcal{L}_6$ obtain the same value for $\sum_{l\in[N]} \delta_l - (\gamma\lambda)^i$ for each $i \neq i^*$. Hence, all ($n - t$ or more) honest parties broadcasts a complaint (or refrain from doing so) in unison. In the former case, all honest parties in $\mathcal{L}_7$ recover $\gamma = \gamma_j$ and $\lambda = \lambda_{j'}$. This can be seen as follows: all parties in $\mathcal{L}_7$ receive $(\gamma^{i,k})_{i\in T_j}$ and $(\lambda^{i,k})_{i\in T_{j'}}$ from each party in $\mathsf{P}_k^6, k \in [n]$. Since there are at most t corrupt parties in $\mathcal{L}_6$, by t robustness of $(\mathsf{Sh}, \mathsf{Rec})$, γ^i computed as $\mathsf{Rec}(\gamma^{i,1}, \ldots, \gamma^{i,n})$ coincides with γ for each i such that P_i^0 is honest. Since there are at least $n - 2t(> t)$ honest parties among $\mathsf{P}_i^0, i \in T_j$, all honest parties correctly recover γ. Similarly, all honest parties recover λ. But then, all honest parties choose $\delta_k = \gamma\lambda$ if $k = 1$ and $\delta_k = 0$ otherwise. Thus, (a) holds in this case.

When all honest parties in $\mathcal{L}_6$ refrain from registering complaints, we will show that $\sum_{l=1}^{N} \delta_l = \gamma\lambda$. Furthermore, each party P_i^7 in the output layer correctly recover δ_k for each k such that $i \in T_k$. These two observations directly imply (a). We justify the first observation as follows: since $|I| = |T_j \cap T_{j'}| \geq t + 1$, there exists $\hat{\imath} \in I$ such that party $\mathsf{P}_{\hat{\imath}}^0$ is honest, and hence correctly carries out VSS. Thus, depending on the value of $\hat{\imath}$, either $\sum_{l=1}^{N} \delta_l = \gamma\lambda$ (when $\hat{\imath} = i^*$) or $(\lambda\gamma)^{\hat{\imath}} = \gamma\lambda$ (when $\hat{\imath} \neq i^*$). In either case, we get $\sum_{l=1}^{N} \delta_l = \gamma\lambda$, since all honest parties in $\mathcal{L}_6$ refrain from registering complaints only if,

$$\sum_{l=1}^{N} \delta_l - (\lambda\gamma)^i = 0, \forall i \in I, i \neq i^*.$$

For each k such that $i \in T_k$, party $\mathsf{P}_i^7, i \in [n]$ correctly receives a CNF share of δ_k from each honest P_j^5 via Future Messaging (see step (ii).2). P_i^7 can correctly reconstruct δ_k from these shares; this follows from CNF VSS being t-robust and the number of corruption is $\mathcal{L}_5$ being at most t. This concludes the proof of (a).

Before proving (b), we informally argue that $\Pi_{j,j'}$ does not leak γ and λ when P_i^0 is honest for each $i \in I$. Observe that, when parties P_i^0 for all $i \in I$ are honest, irrespective of the value of γ and λ,

$$\sum_{l=1}^{N} \delta_l - (\lambda\gamma)^i = 0, \forall i \in I, i \neq i^*.$$

Hence, each honest party P_k^6 refrains from broadcasting a complaint, or broadcasting $(\gamma^{i,k})_{i\in T_j}$ and $(\lambda^{i,k})_{i\in T_{j'}}$. Assume that the statements about security of Π_{FM} and Π_{VSS} hold even when they are composed as in $\Pi_{j,j'}$. Then, the view of $\mathcal{A}$ interacting with $\Pi_{j,j'}$ consists of

1. The shares of λ and γ sent to the corrupt parties in $\mathcal{L}_6$ by parties $\mathsf{P}_i^0, i \in T_j$ and $\mathsf{P}_i^0, i \in T_{j'}$ using Future Messaging in step (i).2 and (i).3, respectively.
2. Let $\mathsf{P}_i^7, i \in C_7$ be the corrupt parties in $\mathcal{L}_7$. Each corrupt party P_i^7 receives CNF shares of δ_k for each k such that $i \in T_k$ from parties in $\mathcal{L}_5$ via Future Messaging initiated in step (ii).2. From this, $\mathcal{A}$ can recover δ_k for each $k \in [N] \setminus \{k^*\}$, where k^* such that $T_{k^*} \subseteq [n] \setminus C_7$. There exists such a $k^* \in [N]$ since $|C_7| \leq t$ and $\{T_1, \ldots, T_N\}$ consists of all subsets of $[n]$ of size $n - t$.
3. View of corrupt parties in $\mathcal{L}_6$ during the secure computation of $\sum_{l=1}^{N} \delta_l - (\lambda\gamma)^i$ for each $i \in I, i \neq i^*$.

Here (1) does not reveal γ and λ since $(\mathsf{Sh}, \mathsf{Rec})$ is a t-secure secret sharing scheme and there are at most t corrupt parties in $\mathcal{L}_6$. Since $\delta_1, \ldots, \delta_N$ are sampled to be additive secret shares of $\gamma\lambda$, $(\delta)_{k\in[N]\setminus\{k^*\}}$ revealed to adversary in (2) are uniformly random irrespective of the value of $\gamma\lambda$. Finally, since each $\mathsf{P}_i^0, i \in I, i \neq I^*$ provides a fresh VSS of $\gamma\lambda$ to $\mathcal{L}_5$ and $\sum_{l=1}^{N} \delta_i = \gamma\lambda$, irrespective of the value of $\gamma\lambda$ and $(\delta_k)_{k\in[N]\setminus\{k^*\}}$, the view of parties in $\mathcal{L}_6$ during secure computation of $\sum_{l=1}^{N}\delta_i - (\lambda\gamma)^i$ is distributed like the CNF VSS of 0, independently, for each $i \in I \setminus \{i^*\}$. Thus, the view of the adversary is identically distributed irrespective of the value of γ and λ, proving (b).

We proceed to formally proving (b) using a hybrid argument. Fix a, b, a', b'. Let $\mathsf{P}_i^7, i \in C_7$ be the corrupt parties in $\mathcal{L}_7$ and let $k^* \in [N]$ such that $T_{k^*} \cap C_7 = \emptyset$. Let Θ be the protocol obtained from $\Pi_{j,j'}$ when (γ,λ) is set to (a,b) in every step of the protocol. Let Θ' be the protocol obtained from $\Pi_{j,j'}$ when (γ,λ) is set to (a',b') in every step of the protocol. We progressively transform Θ to obtain Θ' and argue that the adversary's view is identically distributed across each of these transformations, proving (b).

Transformation $\Theta \rightarrow \Theta_1$. Θ_1 is obtained from Θ by setting γ to a and λ to b in steps (i).2 and (i).3, respectively. As we already observed, since P_i^0 is honest for each $i \in I$, each honest party P_k^{a+6} refrains from broadcasting a complaint, or broadcasting $(\gamma^{i,k})_{i\in T_j}$ and $(\lambda^{i,k})_{i\in T_{j'}}$. Hence, the view of the adversary in Θ and Θ_1 are identical; this uses the same line of argument used for the transformation from Θ to Θ_1 in the proof of Lemma 4.

Transformation $\Theta_1 \rightarrow \Theta'$. Θ' is obtained from Θ_1 by making the following replacements in step (i).1: $\mathsf{P}_{i^*}^0$ verifiably secret shares $\delta_{k^*} - (ab) + (a'b')$ instead of δ_{k^*} (δ_k for $k \neq k^*$ remain unchanged); for each $i \in I, i \neq i^*$, P_i^0 verifiably secret shares $(a'b')$ instead of (ab). Since $(\delta_k)_{k\in[N]\setminus\{k^*\}}$ along with $\delta_{k^*} - (ab) + (a'b')$ forms an additive secret sharing of $(a'b')$, Θ' is $\Pi_{j,j'}$ with (γ,λ) set to (a',b'). By inspecting the protocol, appealing to security of Π_{VSS} and Π_{FM}, it is easy to see that the view of $\mathcal{A}$ is identically distributed. This proves (b) concluding the proof. □

D Layered MPC based on Shamir Secret Sharing

D.1 Details omitted from Section 5.1

Proof of Lemma 6

Lemma 6. *The following properties hold for an execution of Π in the presence of a layered adversary $\mathcal{A}$:*

(a) Let $G \subseteq [n]$ such that P_i^1 is honest if and only if $i \in H_1$. There exist polynomials $\hat{g}(x)$ and $\hat{g}_i(x), i \in H_1$, each of degree at most t, such that $\hat{g}_i(0) = \hat{g}(i)$ and α_i^k output by each honest party P_k^5 coincides with $\hat{g}_i(k)$. Furthermore, if S is honest, $\hat{g}(x) = F(x,0)$.

(b) If S is honest, for any $r, r' \in \mathbb{F}$,

$$\mathrm{ADVR}_{\Pi,\mathcal{A}}(r) \equiv \mathrm{ADVR}_{\Pi,\mathcal{A}}(r').$$

Proof. Suppose S is honest. Then, all the polynomials received by parties in $\mathcal{L}_1$ are consistent; i.e, $f_i(j) = g_j(i)$ for all $i, j \in [n]$. For any $\{i,j\} \in S$, by the correctness of future messaging, $F(i,j) = g_i(j)$ and $F(j,i) = f_i(j)$, whenever P_i^1 is honest. Thus, if $i \in B$ (defined in (iv).3), then P_i^1 is necessarily corrupt. Hence, revealing $f_i(x)$ and $g_i(x)$ for $i \in B$, provide no information to the adversary in addition to what it learned in $\mathcal{L}_1$. By correctness of future messaging, for each $i \in B$, $\hat{g}_i(x)$ and $\hat{f}_i(x)$ recovered in step (v).1 coincide with $f_i(x)$ and $g_i(x)$, respectively. Hence, for any honest P_j^1, by correctness of future messaging, $\hat{f}_i(j) = g_j(i)$ and $\hat{g}_i(j) = f_j(i)$. In conclusion, $i \in B' \cup B$ only if P_i^1 is corrupt. Thus, $|B \cup B'| \leq t$ and, hence, the dealer is not disqualified. Finally, by correctness of future messaging, for each $i \in H_1$, g_i^k is a valid Shamir secret share of $g_i(0)$ whenever P_k^5 is honest, and $g_i(0) = F(i,0)$ for all $i \in H_1$. This proves (a) when S is honest.

Suppose S is corrupt. In step (v).3, if $|B \cup B'| > t$, then (a) holds with $q(x)$ and $g_i(x), i \in H_1$ as zero polynomials. When $|B \cup B'| \leq t$, there exist at least $t+1$ distinct $i \in H_1$ such that $i \notin B \cup B'$.

Define $H_1 \setminus (B \cup B') = H_1'$. Let $i, j \in H_1'$. If $f_i(j) \neq g_j(i)$ or $g_i(j) \neq f_j(i)$, by correctness of Π_{add}, $\{i,j\} \in S$. Furthermore, the purported $F(i,j)$ recovered in step (iv).1 does not coincide with $f_i(j)$ or with $g_j(i)$ if $f_i(j) \neq g_j(i)$. A similar condition holds when $g_i(j) \neq f_j(i)$. Hence, in this case, $\{i,j\} \cap B \neq \emptyset$; a contradiction. Hence, for every $i, j \in H_1'$, the polynomials received by P_i^1 and P_j^1 are pairwise consistent. Additionally, since $H_1' \cap B' = \emptyset$, for every $i \in B$, $\hat{f}_i$ and $\hat{g}_i$ recovered in step (v).1 satisfy $\hat{f}_i(j) = g_j(i)$ and $\hat{g}_i(j) = f_j(i)$ for all $j \in H_1'$. Given these observations, the following claim implies that there exists a unique bivariate polynomial $\hat{F}(x,y)$ with degree at most t in both variables such that, for every $i \in H_1'$, $f_i(x) = \hat{F}(x,i)$ and $g_i(y) = \hat{F}(i,y)$. This claim is proved in [AL17, Claim 5.3].

Claim. Let $f_i(x), g_i(y)$ be polynomials of degree at most t for each $i \in [m]$, where $m > t$. Let $\alpha_k, k \in [m]$ be distinct non-zero elements in $\mathbb{F}$. If for all $i, j \in [m]$, it holds that $f_i(\alpha_j) = g_j(\alpha_i)$, then there exists a unique bivariate polynomial $F(x, y)$ with degree at most t in both variables such that $f_i(x) = F(x, \alpha_i)$ and $g_i(x) = F(\alpha_i, y)$ for all $i \in [m]$.

Finally, we need to argue that if $i \in H_1 \cap (B \cup B')$, then also $f_i(x)$ and $g_i(x)$ are consistent with $\hat{F}(x, y)$. Suppose $i \in H_1 \cap (B \cup B')$ such that $f_i(x) \neq \hat{F}(x, i)$. Since $f_i(x)$ is of degree at most t, there are at most t distinct $j \in H_1'$ such that $f_i(j) = \hat{F}(j, i) = g_j(i)$. In other words, there exists $j \in H_1'$ such that $f_i(j) \neq g_j(i)$. Then $\{i, j\} \in S$; but since $j \notin B$, necessarily, $i \in B$. But then, it must be the case that $\hat{f}_i$ and $\hat{g}_i$ broadcasted in step (v).1 are compatible with $f_j(x), g_j(x)$ for each $j \in H_1'$, otherwise, we get a contradiction. At this point, we conclude that $\hat{g}_i(0) = F(i, 0)$ for all $i \in H_1$. Furthermore, by correctness of future messaging, for each $i \in H_1$, $g_i^k(0)$ is a Shamir share of $\hat{g}_i(0)$ whenever P_k^5 is honest. This proves (a).

We sketch the intuition behind (b). As previously observed, $f_i(x)$ and $g_i(x)$ are never revealed if P_i^1 is honest. Hence, given the security of Π_{add} and Π_{FM}, the adversary only learns the values of $F(x, y)$ on the polynomials that were revealed to corrupt parties in $\mathcal{L}_1$. A formal proof of (b) can be obtained by following a similar line of argument as in proving Lemma 4 (b). We leave this to the reader.

D.2 Random secret sharing, resharing and reinforced secret sharing

Notation. In the sequel, we will use the following notations to denote secret sharing and their manipulations.

1. Let $s \in \mathbb{F}$ and let $f(x)$ be a polynomial of degree at most t such that $f(0) = s$. Suppose s has been secret shared on layer a using $f(x)$; i.e., each $P_i^a, i \in [n]$ gets $s_i = f(i)$. We denote this "state" by $\langle s, f(x) \rangle_a$.
2. We will denote the local addition of shares $\langle a, f(x) \rangle_a$ and $\langle b, g(x) \rangle_a$ by parties in $\mathcal{L}_a$ by $\langle a, f(x) \rangle_a + \langle b, g(x) \rangle_a$. By linearity of Shamir secret sharing, that $\langle a, f(x) \rangle_a + \langle b, g(x) \rangle_a = \langle a+b, g(x)+f(x) \rangle_a$.
3. Similarly, the local multiplication of shares $\langle a, f(x) \rangle_a$ by a constant $\alpha \in \mathbb{F}$ is denoted by $\alpha \langle a, f(x) \rangle_a$. Once again, by linearity, $\alpha \langle a, f(x) \rangle_a = \langle \alpha a, \alpha f(x) \rangle_a$.
4. $\mathcal{L}_a$, holding shares $\langle s, f(x) \rangle_a$, can privately reveal the secret s to a designated P_i^b for $b > a$ by securely communicating all the shares to the party. If $b > a + 1$, this can be realized using future messaging. P_i^b can correctly recover s since the secret sharing is t-robust and future messaging with honest sender can be correctly recovered. We denote this process by $(a)_{\mathsf{P}_i^b} \Leftarrow \langle a, f(x) \rangle$.
5. $\mathcal{L}_a$, holding $\langle s, f(x) \rangle_a$, can reveal s to all parties in layer $b > a$ by communicating all the shares to each $\mathsf{P}_i^b, i \in [n]$. If $b > a + 1$, this can again be realized using future messaging to ensure that parties in layer c, for $a < c < b$, do not learn s. For the same reason as above, all parties in $\mathcal{L}_b$ will correctly recover s. We denote this process by $(s)_b \Leftarrow \langle s, f(x) \rangle_a$.

In this section, we use verifiable secret sharing to implement several useful primitives that we will use extensively in the secure implementation of secure multiplication and MPC. We will develop shorthand notations of the kind defined above for each primitive we define in this section, to facilitate cleaner and shorter representation of these steps in the later protocols.

Random secret sharing. We introduce random secret sharing, in which a random secret is secret shared onto an output layer. This amounts to sampling a polynomial of degree at most t uniformly at random, independent of the view of the adversary, and distributing its shares onto the output layer. This primitive will function as a building block in implement the more complex functionalities we build in this section.

Figure D.1 (Random secret sharing functionality)

PUBLIC PARAMETERS: No input layer, output layer $\mathcal{L}_d, d \geq 6$.
NOTATION: We will denote this functionality by $\langle s, f(x) \rangle_d \leftarrow \$$

Functionality samples $s \leftarrow \mathbb{F}$, and $c_l \leftarrow \mathbb{F}$ for each $1 \leq l \leq t$. Let $f(x) = s + \sum_{l=1}^{t} c_l x^l$. The functionality delivers $f(i)$ to P_i^d for each $i \in [n]$; i.e., distributes $\langle s, f(x) \rangle_d$

Implementing random secret sharing. Using verifiable secret sharing, implementing this protocol is straight forward: We can take assistance from $t+1$ parties (from a previous layer) to verifiably secret share a random secret each onto the output layer. The parties take the sum of these shares as the sampled share. Since at least one amongst $t+1$ of the parties who supplied shares is honest, and parallel VSS is secure, the sum of these shares is guaranteed to be random independent of adversary's view. Given that our VSS protocol consumes 6 layers, random secret sharing with output layer d requires the parties in $d-6$ to supply random secrets. Hence, our resharing protocol works only when the output layer is $\mathcal{L}_6$ or later. This does not pose a limitation since random secret sharing is always used in $\mathcal{L}_6$ or later in all our implementations.

Figure D.2 (A t-secure protocol for random secret sharing)

Public parameters: Output layer $\mathcal{L}_d$ for $d \geq 6$.
Subroutines: A t-secure protocol $\Pi_{\mathsf{ShamirVSS}}$ that implements $f_{\mathsf{ShamirVSS}}$.

1. For each $i \in [t+1]$,
   (a) P_i^{d-6} samples $s_i \leftarrow \mathbb{F}$ and $c_{i,l} \leftarrow \mathbb{F}$ for each $0 \leq l \leq t$, and defines $f_i(x) = s_i + \sum_{l=1}^{t} c_{i,l} x^l$.
   (b) Execute $\Pi_{\mathsf{ShamirVSS}}$ with P_i^{d-6} as dealer to verifiably secret share s_i using $f_i(x)$ onto $\mathcal{L}_d$.
2. Parties in $\mathcal{L}_d$ store $\langle \sum_{i=1}^{t+1} s_i, \sum_{i=1}^{t+1} f_i(x) \rangle_d = \sum_{i=1}^{t+1} \langle s_i, f_i(x) \rangle_d$.

The security of the protocol follows from the above discussion. In fact, the security of parallel invocations of the VSS protocol implies that parallel invocations of random secret sharing remain secure as well.

Theorem 18. *Protocol in Figure D.2 realizes random secret sharing functionality in Figure D.1 with perfect t-security for $t < n/3$.*

All the protocols we construct in the sequel use random secret shares in various ways. For simplicity, we will construct and analyze them assuming that the random secret shares are available as setup. To realize random secret sharing onto a layer using the protocol in Figure D.2, VSS protocols need to be invoked with dealers situated 6 layers above the layer that requires the setup. For now, we overlook this fact and adhere to our convention of having the input client in $\mathcal{L}_0$ with random secret shares as setup made available whenever necessary. This is done to keep the descriptions simple; furthermore, all such protocols are constructed in order to be used as subroutines in the main protocol which implements efficient layered MPC for general function computation given in Figure 5.5. In our final construction, we replace the setup with concurrent protocols securely the setup and argue the security of the ensemble.

Resharing. Going forward, in many protocols, we encounter scenarios where a value that has been verifiably secret shared in a layer needs to be replicated in a later layer. Naively duplicating the same secret sharing by sending the shares to the later layer using future messaging is clearly not secure. The adversary can corrupt t parties each in both the layers and learn $2t$ shares of the secret, breaking security. The later layer needs to necessarily receive a fresh resharing of the same value. The resharing functionality allows parties in $\mathcal{L}_a$ with (a valid) secret sharing of a secret s to "handover" the secret to parties in $\mathcal{L}_b$, for any $b > a$, by providing a fresh secret sharing of s. The following functionality requires that the input clients hold a valid Shamir secret sharing, and that at most t input clients are corrupt.

Figure D.3 (Resharing functionality)

Public parameters: Output layer $\mathcal{L}_d$ for any $d \geq 1$.
Secret inputs: $\langle s, f(x) \rangle_0$.
Notation: $\langle s, f'(x) \rangle_d \leftarrow \langle s, f(x) \rangle_0$.

1. Input clients reveal $\langle s, f(x) \rangle_0$ to the functionality who reconstructs s. The reconstruction is correct since at most t parties in $\mathcal{L}_0$ are corrupt.
2. Functionality samples $c_l \leftarrow \mathbb{F}, 1 \leq l \leq t$, defines $f'(x) = s + \sum_{i=1}^{t} c_l$, and distributes $\langle s, f'(x) \rangle_d$.

Implementing resharing. For implementing resharing, we use a setup in which t random secrets are secret shared onto the input clients. The protocol works as follows:

Figure D.4 (A t-secure implementation of resharing)

PUBLIC PARAMETERS: Output layer $d \geq 1$.
SECRET INPUTS: $\langle s, f(x)\rangle_0$.
SETUP: $\langle \alpha_l, f_l(x)\rangle_0 \leftarrow \$$ for $1 \leq l \leq t$.
OUTPUT: $\langle s, f'(x)\rangle_d$, where $f'(x) = s + \sum_{l=1}^{t} \alpha_l x^l$.

1. For each $i \in [n]$, execute

$$(s + \sum_{l=1}^{t} i^l \alpha_l)_{\mathsf{P}_i^d} \Leftarrow \langle s, f(x)\rangle_0 + \sum_{l=1}^{t} i^l \langle \alpha_l, f_l(x)\rangle_0$$

   Recall that this involves the following steps:
   (a) For each $j \in [n]$, let $s_j = f(j)$ be the share of s held by P_j^0. For each $1 \leq l \leq t$, let $\alpha_{l,j} = f_l(j)$ be the share of α_l held by P_j^0. Then, P_j^0 sends $s_j + \sum_{l=1}^{t} i^l \alpha_{l,j}$ to P_i^d using future messaging.
   (b) For each $j \in [n]$, P_i^d recovers $s'_{i,j}$ as the output of future messaging with P_j^0 as sender. P_i^d stores $\mathsf{Rec}(s'_{i,1}, \ldots, s'_{i,n})$ as their share of s.
   Whenever P_j^0 is honest, $s_{i,j} = g(i)$, where $g(x) = f(x) + \sum_{l=1}^{t} i^l f_l(x)$. Since at most t parties are corrupt, P_j^0 correctly recovers $s_i = g(0) = f'(i)$, where $f'(x) = s + \sum_{l=1}^{t} \alpha_l x^l$.
2. Each P_i^d stores $s_i = s + \sum_{l=1}^{t} i^l \alpha_l$ as their (re)share of s.

The correctness of the above protocol is clear from the description. Since $\langle \alpha_l, f_l(x)\rangle_0 \leftarrow \$$ for $1 \leq l \leq t$ are randomly sampled, by linearity of Shamir secret sharing and security of future messaging, the shares received by corrupt output clients are identically distributed irrespective of s and $f(x)$. Hence, the view of an adversary can be simulated even if it knows $s, f(x)$. Observe that, assuming the setup, the protocol necessarily implies parallel future messaging. Since our future messaging protocol is secure when executed in parallel, the resharing protocol also remains secure under parallel composition.

Theorem 19. *Protocol in Figure D.4 realizes the resharing functionality in Figure D.3 with perfect t-security for $t < n/3$.*

Reinforced secret sharing. We next define an enhanced form of Shamir secret sharing that we will refer to as reinforced secret sharing. This notion is defined in [CDN15] as *verifiable secret sharing*, which we defined in differently. We will use reinforced secret shares for securely processing the gates during the circuit evaluation phase of our MPC protocol.

Definition 10. *A (t, n)-reinforced secret sharing of $s \in \mathbb{F}$ consists of the following $(n + 1)$ distinct Shamir secret shares:*

1. *Sample a polynomial $f(x)$ of degree at most t uniformly at random under the constraint $f(0) = s$.*
2. *For each $i \in [n]$, sample a polynomial $f_i(x)$ of degree at most t uniformly at random under the constraint $f_i(0) = f(i)$.*
3. *Distribute shares $\langle s, f(x)\rangle$ and $\langle f(i), f_i(x)\rangle$.*

Reconstruction amounts to applying the reconstruction algorithm for Shamir secret sharing on shares $\langle s, f(x)\rangle$.

We will denote a reinforced secret sharing of a secret s using $f(x), (f_i(x))_{i\in[n]}$, as defined above, by $[[s, f(x), (f_i(x))_{i\in[n]}]]$.

In our constructions, we build a reinforced secret sharing of a secret from a valid Shamir secret sharing of the same. This notion is formalized by the reinforced resharing functionality described below. The functionality requires that the input clients hold a valid Shamir secret sharing, and that at most t input clients are corrupt.

Figure D.5 (Reinforced resharing functionality)

Public parameters: Output layer $\mathcal{L}_d$ for any $d \geq 1$.
Secret inputs: $\langle s, f(x)\rangle_0$.
Notation: $[[s, f'(x), (f'_i(x))_{i\in[n]}]]_d \leftarrow \langle s, f(x)\rangle_0$.

1. Input clients ($\mathcal{L}_0$) reveal $\langle s, f(x)\rangle_0$ to the functionality who recovers s. The reconstruction is correct since at most t parties are corrupt.
2. Functionality samples a random polynomial $f'(x)$ of degree at most t conditioned on $f'(0) = s$; for each $i \in [n]$, it samples a random polynomial $f'_i(x)$ of degree at most t conditioned on $f'_i(0) = f'(i)$.
3. Functionality distributes $\langle s, f'(x)\rangle_d$, and $\langle f'(i), f'_i(x)\rangle_d$ for each $i \in [n]$.

Implementing reinforced resharing. Our protocol works as follows: First, the secret s is reshared to the output clients. This involves sampling secret shares of random secrets α_l for each $1 \leq l \leq t$ and delivering $s_i = f(i)$ to output client i, where $f(x) = s + \sum_{l=1}^{t} \alpha_l x^l$. This is realized exactly as in our implementation of resharing (Figure D.4). Observe that the input clients possess a secret sharing of s_i for each i as well, indeed, s_i was revealed to output client i by revealing the shares of s_i. But then, each s_i can be reshared onto the output layer using the resharing protocol. This achieves reinforced resharing of s. The security of the construction follows directly from the security of resharing. Similar to the resharing protocol, reinforced resharing only uses parallel invocations of future messaging protocol; hence, it remains secure under parallel composition.

Figure D.6 (A t-secure implementation of reinforced resharing)

Public parameters: Output layer $d \geq 1$.
Secret inputs: $\langle s, f(x)\rangle_0$.
Setup: $\langle \alpha_l, g_l(x)\rangle_0 \leftarrow \$$ for $1 \leq l \leq t$ and $\langle \alpha_{i,l}, g_{i,l}(x)\rangle_0 \leftarrow \$$ for $i \in [n]$ and $1 \leq l \leq t$.
Subroutines: A t-secure protocol Π_{reshare} that implements the resharing functionality.
Output: $[[s, f'(x), (f'_i(x))_{i\in[n]}]]_d$.

1. For each $i \in [n]$, define
$$\langle s_i, g'_i(x)\rangle_0 = \langle s, f(x)\rangle_0 + \sum_{l=1}^{t} i^l \langle \alpha_l, g_l(x)\rangle_0.$$
Here, $f'(x) = s + \sum_{l=1}^{t} \alpha_l x^l$ and $s_i = f'(i)$ for all $i \in [n]$.
2. For each $i \in [n]$, execute $(s_i)_{\mathsf{P}_i^d} \Leftarrow \langle s_i, g'_i(x)\rangle_0$ (See Figure D.3 step 1).
3. For each $i \in [n]$, reshare $\langle s_i, g'_i(x)\rangle_0$ using Π_{reshare} using $\langle \alpha_{i,l}, g_{i,l}(x)\rangle_0$ for $1 \leq l \leq t$ as setup. For each $i \in [n]$, this achieves
$$\langle s_i, f'_i(x)\rangle_d \leftarrow \langle s_i, g'_i(x)\rangle_0, \text{ where } f'_i(x) = f'(i) + \alpha_{i,1}x^1 + \ldots + .\alpha_{i,t}x^t.$$
4. Parties in $\mathcal{L}_d$ store $[[s, f'(x), (f'_i(x))_{i\in[n]}]]_d$.

Theorem 20. *Protocol in Figure D.4 realizes the reinforced resharing functionality in Figure D.3 with perfect t-security for $t < n/3$.*

D.3 Details omitted from Section 5.2

We define the functionality for multiplication with helper in Figure D.7.

Figure D.7 (Multiplication with helper functionality)

Public parameters: Helper is P_1^0 and output layer $\mathcal{L}_9$.
Input: $\langle\alpha, f_0(x)\rangle_0, \langle\beta, g_0(x)\rangle_0$; helper P_1^0 holds α, β.

1. Input clients ($\mathcal{L}_0$) reveal the shares of α and β to the functionality, who reconstructs α, β. Additionally, P_1^0 sends γ to the functionality.
2. If $\gamma = \alpha\beta$, functionality distributes a secret sharing of γ onto $\mathcal{L}_3$; i.e., $\langle\alpha\beta, h(x)\rangle_9$, where $h(x)$ is a random polynomial of degree at most t conditioned on $h(0) = \alpha\beta$. Otherwise, functionality delivers $\perp$ to all parties.

The protocol for multiplication with helper is formally described in Figure D.8.

Protocol D.8 (t-securely realizing multiplication with helper)

Public parameters: Input layer $\mathcal{L}_0$, helper P_1^0, output layer $\mathcal{L}_8$.
Inputs: $\langle\alpha, f_0(x)\rangle_0, \langle\beta, g_0(x)\rangle_0$; helper P_1^0 holds α, β.
Subroutines: $\Pi_{\mathsf{ShamirVSS}}$ and Π_{Reshare} implementing verifiable secret sharing and resharing functionality.
Setup: Sufficiently many random secret shares in $\mathcal{L}_0$ and $\mathcal{L}_6$ required to execute resharing.

1. P_1^0 samples $\alpha_l \leftarrow \mathbb{F}$, $\beta_l \leftarrow \mathbb{F}$ for each $1 \leq l \leq t$. Define $f(x) = \alpha + \sum_{l=1}^{t} \alpha_l x^l$ and $g(x) = \beta + \sum_{l=1}^{t} \beta_l x^l$. Let $f(x)g(x) = \sum_{l=0}^{2t} \gamma_l x^l$. Use $\Pi_{\mathsf{ShamirVSS}}$ with P_1^0 as dealer and $\mathcal{L}_6$ as shareholders, to distribute shares
$$\langle\alpha_l, f_l(x)\rangle_6, \forall 1 \leq l \leq t \quad \langle\beta_l, g_l(x)\rangle_6, \forall 1 \leq l \leq t \quad \langle\gamma_l, h_l(x)\rangle_6, \forall 0 \leq l \leq 2t$$
Finally, execute Π_{Reshare} to realize
$$\langle\alpha, f_0'(x)\rangle_6 \leftarrow \langle\alpha, f_0(x)\rangle_0 \qquad \langle\beta, g_0'(x)\rangle_6 \leftarrow \langle\beta, g_0(x)\rangle_0$$
2. For each $i \in [n]$, reveal the following linear combinations of shares to P_i^7:
$$(\hat{f}(i))_{\mathsf{P}_i^7} \Leftarrow \langle\alpha, f_0'(x)\rangle_6 + i\langle\alpha_1, f_1(x)\rangle_6 + \ldots + i^t\langle\alpha_t, f_t(x)\rangle_6$$
$$(\hat{g}(i))_{\mathsf{P}_i^7} \Leftarrow \langle\beta, g_0'(x)\rangle_6 + i\langle\beta_1, g_1(x)\rangle_6 + \ldots + i^t\langle\alpha_t, f_t(x)\rangle_6$$
$$(\hat{h}(i))_{\mathsf{P}_i^7} \Leftarrow \langle\gamma_0, h_0'(x)\rangle_6 + i\langle\gamma_1, h_1(x)\rangle_6 + \ldots + i^{2t}\langle\gamma_{2t}, h_t(x)\rangle_6$$
Here, $\hat{f}(x) = \alpha + \sum_{l=1}^{t} \alpha_l x^l$, $\hat{g}(x) = \beta + \sum_{l=1}^{t} \beta_l x^l$ and $\hat{h}(x) = \sum_{l=0}^{2t} \gamma_l x^l$. Further, execute Π_{Reshare} to realize $\langle\gamma_0, h(x)\rangle_9 \leftarrow \langle\gamma_0, h_0\rangle_6$.
3. Each $\mathsf{P}_i^7, i \in [n]$ checks if $\hat{f}(i)\hat{g}(i) = \hat{h}(i)$. Otherwise, broadcast a complaint.
4. For each $i \in [n]$, if P_i^7 registered a complaint, execute public reveal as follows:
$$(\hat{f}(i))_9 \Leftarrow \langle\alpha, f_0'(x)\rangle_6 + i\langle\alpha_1, f_1(x)\rangle_6 + \ldots + i^t\langle\alpha_t, f_t(x)\rangle_6$$
$$(\hat{g}(i))_9 \Leftarrow \langle\beta, g_0'(x)\rangle_6 + i\langle\beta_1, g_1(x)\rangle_6 + \ldots + i^t\langle\alpha_t, f_t(x)\rangle_6$$
$$(\hat{h}(i))_9 \Leftarrow \langle\gamma_0, h_0'(x)\rangle_6 + i\langle\gamma_1, h_1(x)\rangle_6 + \ldots + i^{2t}\langle\gamma_{2t}, h_t(x)\rangle_6$$
Note that, $\hat{f}(i), \hat{g}(i)$ and $\hat{h}(i)$ are to be revealed only for $i \in [n]$ with a registered complaint. Since the complaints are available in $\mathcal{L}_8$, this can be achieved by having each P_i^6 secret share their share of $\hat{f}(i)$ and so on, onto $\mathcal{L}_8$ and then having $\mathcal{L}_8$ selectively reveal these shares to P_i^9 only for $i \in [n]$ with a complaint.
5. For each $i \in [n]$ with a complaint, all parties in $\mathcal{L}_9$ check if $\hat{f}(i)\hat{g}(i) = \hat{h}(i)$ If the equality check succeeds for all complaints, then the parties store $\langle\gamma_0, h(x)\rangle_9$ as shares of $\alpha\beta$.