Perfect MPC over Layered Graphs

Bernardo David[1(✉)], Giovanni Deligios[2], Aarushi Goel[3], Yuval Ishai[4], Anders Konring[1], Eyal Kushilevitz[4], Chen-Da Liu-Zhang[3], and Varun Narayanan[4]

[1] IT University of Copenhagen, Copenhagen, Denmark
`beda@itu.dk`
[2] ETH Zurich, Zürich, Switzerland
[3] NTT Research, Sunnyvale, USA
[4] Technion - Israel Institute of Technology, Haifa, 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

This paper is a merged version of the papers [19,20].

© International Association for Cryptologic Research 2023
H. Handschuh and A. Lysyanskaya (Eds.): CRYPTO 2023, LNCS 14081, pp. 360–392, 2023.
https://doi.org/10.1007/978-3-031-38557-5_12

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 [4] (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 [41] 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 [41], 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 [2,3,22,34] explored more efficient protocols with other security guarantees under further restrictions on the mobile adversary, but still fell short of 1-round epochs or achieving the optimal corruption threshold ($t < n/3$) of BGW.

Departing from player replaceability[1] and anonymous committees of distributed ledgers, the notion of You Only Speak Once (YOSO) MPC (introduced in [28]) 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 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, makes 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 [28] 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 [13] is a variant of this model without target-anonymous channels, where parties may act in more than one round before being substituted, but the results presented in [13] fall short of full security, as they do not achieve G.O.D. Another variation was shown in SCALES [1], 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.

[1] A term from [30] for protocols where a new set of parties executes each round.

Inspired by the original mobile adversary characterized by [41] and the recent line of work on MPC with dynamic committees [1,13,28], we ask again the question originally settled in BGW [4] 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 of 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 [8,31] and Universal Composability [9].

Layered MPC is similar to maximally-Fluid MPC [13] with parties only executing one round. We show that a secure layered MPC protocol is also secure against a maximally mobile adversary [41], that moves after every round. In comparison to YOSO [28], 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, similar to Fluid MPC, 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 Sect. 3 we construct basic primitives that help realize layered VSS based on CNF[2] (replicated) secret sharing. We present a nontrivial adaptation of a VSS protocol of Gennaro et al. [27] 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 their 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*

[2] 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 $i \notin T$.

$d = O(1)$ 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.*

In Sect. 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 [17], in the full version of this paper [19] 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 Sect. 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.*

Further, in Sect. 6, we present a computationally secure, efficient layered protocol that achieves G.O.D. against adversaries who can corrupt $t < n/2$ parties in each layer.

Theorem 4 (Efficient Layered MPC for $t < n/2$). *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 $t < n/2$, there is an $(n, t, O(D))$-layered MPC protocol for f assuming non-interactive linearly-homomorphic equivocal commitments. The communication complexity is $M \cdot O(n^9)$ field elements over the point-to-point channels and $M \cdot O(n^5)$ field elements + $M \cdot O(n^{10} \cdot \lambda)$ bits over the broadcast channels, where λ is the security parameter.*

Proactive MPC. The original concept of proactive MPC put forward by [41] considered an adversary that has the ability to corrupt a fresh set of parties in *every* round of the protocol. We refer to such an adversary as maximally mobile. This notion is formally defined in the full version of the paper [19], 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. We also remark on an alternate and stronger notion of maximal adversary in the full version [19, Remark 2], against which perfectly secure VSS and MPC are impossible with the optimal threshold of $t < n/3$ corruptions in each layer. This allows us to extend our security analysis from the layered to the proactive setting. The full version [19] defines maximally Proactive Secret Sharing and MPC and we obtain the following threshold-optimal result by combining Theorem 3 and [19, 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 [28] 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 Sect. 3.1 we provide a thorough analysis of an important primitive in layered MPC called *Future Messaging*. The functionality f_{FM} is described in Sect. 3.1 and presented in [19, Figure 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 5 (Restatement of Theorem 7). *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 of Future Messaging linear in d.

The layered model allows for layer-to-layer broadcast: 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 deterministic broadcast in the setting of layered MPC is possible only if $t = 0$. Our analysis is shown in the full version [19], where we cast the result of [26] to the setting of layered MPC and obtain the following result.

Theorem 6. *Deterministic perfect Broadcast in the setting of layered MPC is possible iff* $t = 0$.

This limitation can be overcome by the use of randomization. Several works achieve broadcast in the honest-majority setting with overwhelming probability after a number of rounds that is linear in the security parameter, without setup tolerating $t < n/3$ corruptions [23], and with different types of setup tolerating $t < n/2$ corruptions [24,25,29,36].

These protocols can be ported to the layered setting at the cost of decreasing the corruption threshold by a factor that is linear in the security parameter. This is done by naively porting the protocol to the layered setting after ensuring that the parties 'persist' across all the layers by simply forwarding the view of each party to their counterpart in the next layer. When the adversary corrupts t' parties in each layer, by the end of the protocol, the adversary would corrupt at most $t = t' \cdot O(\kappa)$ parties executing different party roles, for security parameter κ. If the total number of corruptions that the original protocol tolerates is bounded by $t < n/3$ (resp. $t < n/2$), we have that the ported protocol remains secure. Obtaining the optimal corruption threshold $t' < n/3$ without setup, or $t' < n/2$ with setup, for broadcast is beyond the scope of this paper.

1.2 Related Work

We summarize the relationship between previous works in similar settings and our results in Table 1. We discuss further related works below.

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 [2,3,11,34]. 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 [21,22,44]. Finally, proactive techniques in asynchronous settings have been treated in [7,43].

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 notion of player replaceability (where the set of parties get replaced in every round) has previously been studied in the context of consensus primitives [6,12,40,42]. The recent focus on this setting spurred new interest in (dynamic) proactive techniques [32,38]. Particularly interesting, is the

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)

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	[5]	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	[28] (YOSO)	statistical	full (w/setup†)	$\mathsf{poly}(n)$	$t < n/2^*$
	[13] (Fluid)	statistical	w/abort	$\mathsf{poly}(n)$	$t < n/2$
	[41]	perfect	full	$\mathsf{poly}(n)$	$t < n/d$
	Section 4.4	perfect	full	$2^{O(n)}$	$t < n/3$
	Section 5	perfect	full	$\mathsf{poly}(n)$	$t < n/3$
	Section 6	computational	full	$\mathsf{poly}(n)$	$t < n/2$

definition of evolving committee secret sharing [5] 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 [1,13,28,30] 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 [28], 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 [28, 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 [13] 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[3]. Finally, a crucial difference is that the known protocols 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 [33], 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

[3] The inherent issue with state complexity originates from a common misconception (see fx [18]) that *any* general arithmetic circuit can be transformed into a layered circuit with same depth and only linear overhead in width.

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 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 Sect. 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 [8] 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. 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 a random 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 [16]. 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 full version [19, Fig. 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 $\mathcal{L}_i$, $0 \leq i \leq d$, where $|\mathcal{L}_i| = n$. Parties in the first layer $\mathcal{L}_0$ and the last layer $\mathcal{L}_d$ are referred to as* input clients *and* output clients*, respectively.*
- **Interaction pattern.** *The interaction consists of d rounds, where in round i parties in $\mathcal{L}_{i-1}$ may send messages to parties in $\mathcal{L}_i$ over secure point-to-point channels. 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*[4] *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 [8, 31, 35]. We require* perfect full security *(with guaranteed output delivery).*

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.

[4] In the coming sections our security analysis is with respect to non-adaptive adversaries for simplicity. In Sect. 2.2 we justify this leap appealing to the work of [10].

The Need for Ideal Broadcast: In the full version [19, Appendix 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 the [19, Section 2.2] we precisely define maximally proactive security and prove that it is implied by layered MPC. Also see Remark 2 in the section that addresses a natural and stronger notion of maximally proactive security.

2.2 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 [8] 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 [8, 31]. 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 [37] and argue that the protocol is, in fact, secure under the definition of universal composability[5].

[5] 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 [8, 31].

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 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. [10] 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.

3 Basic Primitives

We introduce the basic primitives Future Messaging (f_{FM}) and Multiparty Addition (f_{add}) 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[6] 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 formalized in full version [19, Figure 3.1].

[6] 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.

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 the full version [19] for more details.

We capture the functionality realized by parallel execution of our future messaging protocol using a corruption aware functionality in Fig. 3.1.

Figure 3.1 (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.

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 $\ell = \lfloor \frac{d}{2} \rfloor$. 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 $\mathcal{L}_d$.

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_{FM}^n (Fig. 3.1) from $\mathcal{L}_0$ to $\mathcal{L}_d$ using f_{FM}^n from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ and from $\mathcal{L}_\ell$ to $\mathcal{L}_d$. The protocol is formally described in Fig. 3.2.

Figure 3.2 ($\mathit{\Pi}_{\mathsf{FM}}^n$, an (n, t, d)-layered protocol realizing f_{FM}^n)

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_{FM}^n (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 $\mathsf{P}_k^\ell \in \mathcal{L}_\ell$ in f_{FM}^n from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ to $\left(s_{(i,1),k}, \ldots, s_{(i,n),k}\right)$.
3. Each party $\mathsf{P}_k^\ell : k \in [n]$ receives $(\hat{s}_{(i,1),k}, \ldots, \hat{s}_{(i,n),k})$ from $\mathsf{S}_i, i \in [n]$ (delivered by f_{FM}^n).
4. $\mathsf{P}_k^\ell, k \in [n]$ sets the message to $\mathsf{R}_j \in \mathcal{L}_d$ in f_{FM}^n 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 $\mathsf{Rec}(\hat{s}_{(i,j),1}, \ldots, \hat{s}_{(i,j),n})$.

Lemma 1 (Layered protocol for f_{FM}^n). *Let* $(\mathsf{Sh}, \mathsf{Rec})$ *be a robust* (t, n) *secret-sharing scheme [19, Definition 6], the* (n, t, d)*-layered protocol in Fig. 3.2 realizes the functionality* f_{FM}^n *in Fig. 3.1 with perfect security for* $t < n/3$.

We formally describe the simulator and provide a formal proof in the full version [19].

Going forward, we will focus on the (non-parallel) Future Messaging functionality f_{FM} [19, 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_{FM}^n $(n = 1)$ and a protocol was outlined informally in the beginning of this section.

Theorem 7. *For any* $d > 0$*, and message domain* M*, there exists an* (n, t, d)*-layered protocol* $\mathit{\Pi}_{\mathsf{FM}}$ *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 $\ell = \left\lfloor \frac{d}{2} \right\rfloor$. 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 Fig. 3.2, 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 $\mathcal{L}_1$. □

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 $m, m' \in M$,

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

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 [19, Figure 3.4] and can be realized by an $(n, t, 2)$-layered protocol as outlined below.

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 the full version [19]. 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 2. *The following statements hold when an (n, t, d)-layered protocol realizing f_{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 Sect. 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[7] but is substantially harder to realize. The functionality is described in [19, Figure 4.1].

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 3.

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.

[7] 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 [28].

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 layered adversary if all receivers are honest. Finally, by (1), 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 the full version [19]. 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 3 and a formal proof is provided in [19].

Lemma 3. *For any $T \in \mathcal{T}$, the following properties hold for any weak future multicast protocol 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 the protocol. 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 8. *There is a secure* $(n, t, 5)$*-layered protocol realizing future multicast with input client* S *and output clients in* R*.*

Proof. Let $\Pi_{\mathsf{weak\text{-}FMcast}}$ be a protocol realizing weak future multicast. By statement (a) in Lemma 3, 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, at the end of the future multicast protocol, say Π_{FMcast}, the outputs of all receivers are the same 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 3, 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 the full version [19].

4.2 Verifiable Secret Sharing

Using future multicast presented in Sect. 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 Fig. 4.1 realizes VSS from a dealer in $\mathcal{L}_0$ to shareholders in $\mathcal{L}_5$.

Figure 4.1 (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 Fig. 4.1 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 4.2 (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 9. *The protocol in Fig. 4.1 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 the full version of the paper [19]. □

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 f_{mult} (presented in [19, Figure 4.5]) takes valid CNF secret shares of two values r and s and computes 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. In contrast, by default, a layered adversary is allowed to corrupt arbitrarily many input and output clients.

Implementing f_{mult}. Suppose $r_1, \dots, r_n$ and $s_1, \dots, s_n$ are CNF secret shares of two values r and s, respectively. Recall that, when $\mathcal{T} = \{T_1, \dots, T_N\} = \{T \subset [n] : |T| = n - t\}$, 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.

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 [39] is provided in [19, Figure C.1]. 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 [19, Figure 4.6].

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 the full version of the paper [19, Section C.5].

Lemma 4. *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 4, 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 4, 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. We obtain the following theorem.

Theorem 10. *There is an $(n, t, 7)$-layered protocol realizing f_{mult}, for $t < n/3$.*

Executing the above protocol 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 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 the full version [19, Figure 4.7]. We get the following result.

Theorem 11. *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 $L_1, \dots, L_D$. Then, for any $t < n/3$, there is an $(n, t, 6 + 7D)$-layered MPC protocol for f.*

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 [15, 16] 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 [19, Figure 5.1].

Implementing $f_{\mathsf{ShamirVSS}}$. The layered protocol realizing $f_{\mathsf{ShamirVSS}}$ is provided in the full version [19, Figure 5.2]. We sketch the 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 [19, 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 5.

Lemma 5. *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 $r, r' \in \mathbb{F}$,

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

Using Π as a subroutine, verifiable secret sharing is achieved as follows (described in [19, Figure 5.2]). Let $q(x) = c_0 + c_1x + \ldots c_tx^t$ 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 $\mathsf{P}_i^5, i \in H_5$ are the set of honest parties in $\mathcal{L}_5$. By Lemma 5, 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_5^k holds $\alpha_{l,i}^k = \hat{g}_{l,i}(k)$. Since $|H_5| \ge n - t$, each $\mathsf{P}_j^6, j \in [n]$ recovers $\gamma_{i,j} = \mathsf{Rec}(\alpha_{l,j}^1, \ldots, \alpha_{l,j}^n) = \sum_{l=0}^t \hat{g}_l(i)j^l$ for all $i \in H_1$. Hence, $\gamma_j = \mathsf{Rec}(\gamma_{1,j}, \ldots, \gamma_{n,j}) = \sum_{l=0}^t \hat{g}_l(0)j^l$. Defining $\hat{q}(x) = \hat{g}_l(0)x^l$, we conclude that each P_j^6 receives $\hat{q}(j)$ as required in verifiable Shamir secret sharing. When S is honest, by Lemma 5 (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 5 (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 $\mathsf{P}_i^6, i \in C_6$ are the set of corrupt parties in $\mathcal{L}_6$.

In Protocol [19, Figure 5.2], the polynomial secret shared in $\mathcal{L}_6$ is exclusively determined by $\alpha_{l,i}^k$, for $i \in [n]$ and $0 \le l \le t$ stored by the honest parties P_k^5. In other words, the dealer is committed to polynomial $\hat{g}_l(x), 0 \le l \le t$ (as described in Lemma 5) when all the honest parties in $\mathcal{L}_5$ finish receiving messages from their predecessors. Furthermore, by Lemma 5, 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 $\mathsf{P}_i^0, i \in [n]$ as dealer and $\mathcal{L}_6$ as shareholders, we realize a parallel VSS functionality with t-security.

Theorem 12. *There is an $(n, t, 6)$-layered protocol which, when executed in parallel, realizes parallel Shamir-VSS for $t < n/3$ 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 [16] 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 [19, Figure D.1] and its implementation in [19, Figure D.2]). The resharing functionality is formally defined in [19, Figure D.3], and it is implemented as outlined above in [19, 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 [16], 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 [16]. 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 $f(x) = \alpha + \sum_{l=1}^{t} \alpha_l x^l, g(x) = \beta + \sum_{l=1}^{t} \beta_l x^l$, and $h(x) = \sum_{l=0}^{2t} \gamma_l x^l$. 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 $\mathsf{P}^7_i, i \in [n]$. Each P^7_i 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 part of the 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 [19, Figure D.8].

Theorem 13. *There is a layered protocol that realizes multiplication with helper functionality 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 shared in the same layer. Hence, we need each $f(i)$ and $g(i)$ to be further secret shared onto the input layer. We refer to the 'data structure' where each share of a value is further verifiably secret shared as reinforced secret sharing (formally descrived in [19, Definition 10]). The multiplication functionality takes *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.

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, which takes valid Shamir shares of a value from the input clients and distributes a randomly sampled reinforced secret sharing of the same value. This functionality is formally described in [19, Figure D.6], and implemented (along the lines of Shamir resharing) in [19, Figure 5.4].

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

Theorem 14. *There is a layered protocol that realizes multiplication functionality 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. The protocol is formally described in [19, Figure 5.5].

Theorem 15. *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.

6 Computational Efficient Layered MPC for $t < n/2$

We introduce a computationally-secure layered MPC protocol with guaranteed output delivery, based on (non-interactive) equivocal linearly homomorphic commitments. We give a high-level overview and defer details to the full version [20].

Future Messaging. The primitive is achieved similarly as its perfectly-secure counterpart, but to tolerate $t < n/2$ corruptions, we cannot rely on plain error correction. Instead, parties broadcast commitments to (coefficients of) the polynomials used to share their values to the future layers. Every time a party wishes to re-share an intermediate value, they re-use the commitment to the constant coefficient, thereby ensuring that the proper value is being re-shared.

Distributed Commitments. This primitive (also referred to as *weak* secret sharing [27]) allows a dealer $\mathsf{D} \in \mathcal{L}_c$ to commit to a value s towards a future layer $\mathcal{L}_{c'}$, and later open the original value towards another further layer $\mathcal{L}_{c''}$. If D is honest, the opened value is s, and no information about s is revealed before the opening phase. Moreover, even if D is corrupted, the commit phase uniquely determines value s', such that the opening phase can only output s' or $\perp$.

The dealer $\mathsf{D} \in \mathcal{L}_c$ samples random degree-t polynomials $f(x), r(x)$, such that $f(0) = s$, computes a commitment to each coefficient in f using the coefficients in r as randomness and broadcasts these commitments to the future layers. The dealer then sends the evaluation points $(s(i), r(i))$ using future messaging to party $P_i^{c'}$. To reconstruct, layer $\mathcal{L}_{c'}$ broadcast these pairs to the future layers, and each party $P_i^{c''} \in \mathcal{L}_{c''}$ checks for each received pair whether it is consistent with the corresponding commitment. If there are more than t consistent pairs, interpolate a degree-t polynomial $f'(x)$ and output $f'(0)$. Otherwise, output $\perp$.

Remark 2. We can achieve a distributed commitment that allows to commit *to the same value* towards separate layers $\mathcal{L}_{c'}$ and $\mathcal{L}_{c''}$, such that even if P_d^c is corrupted, there exists a unique value s' such that the value that is opened by either layer is s' or $\perp$: let the dealer P_d^c execute the above protocol towards layers $\mathcal{L}_{c'}$ and $\mathcal{L}_{c''}$ with polynomials $f(x)$ and $f'(x)$ such that $f(0) = s = f'(0)$, but using *the same* commitment for the constant coefficient.

Verifiable Secret Sharing. For VSS, the dealer $\mathsf{D} \in \mathcal{L}_c$ with input s, samples random degree-t polynomial $f(x)$ with $f(0) = s$, and (duplicate) commits to each coefficient of f towards layers $\mathcal{L}_{c_1}$ and $\mathcal{L}_{c'}$ using the distributed commitment. This results in a matrix $\mathbf{M} = [{}_{,i,j}]_{0 \leq i,j \leq t}$ of public commitments, where ${}_{,i,j}$ is a commitment to the j-th coefficient of the polynomial used to share f_i: by linearity of the commitment, parties implicitly hold commitments to each evaluation $f(i)$. Using future messaging, the dealer D sends $s_i = f(i)$ and its opening information to $P_i^{c_1}$. Party $P_i^{c_1}$ can check that the received information is consistent with the published commitments, and broadcast to future layers a complaint if the check does not succeed. If the check succeeds, $P_i^{c_1}$ commits to s_i towards layer $\mathcal{L}_{c'}$; to ensure that $P_i^{c_1}$ commits to the value they received from P_c^d, the party uses the commitment to the constant term that is implicit from the published commitments in $\mathbf{M}$. If a complaint was raised by $P_i^{c_1}$, parties in layer $\mathcal{L}_{c'}$ publicly open s_i (and if the opened value is $\perp$, the dealer is disqualified). To reconstruct: for each index i corresponding to a party that did not complain, parties jointly reconstruct s_i. The final layer $\mathcal{L}_{c''}$ then uses any $t+1$ reconstructed shares to interpolate the secret. Moreover, as in Remark 2, by having both D and parties in layer $\mathcal{L}_{c_1}$ duplicate distribute commitments of s_i for all $i \in [1, n]$ towards layers $\mathcal{L}_{c'}$ and $\mathcal{L}_{\tilde{c}}$ for some $\tilde{c} \geq c'$, one guarantees that if D is not disqualified, then both $\mathcal{L}_{c'}$ and $\mathcal{L}_{\tilde{c}}$ hold sharings of the *same* value.

2-Level Verifiable Secret Sharing. To simplify the description of the MPC protocol, it is helpful that each party holds as part of their state a Shamir share of each wire value. For that, we modify the VSS as follows: the dealer D uses the above duplicate VSS to distribute shares of coefficients of f towards layers $\mathcal{L}_{c_1}$ and $\mathcal{L}_{c'}$, where $f(x)$ is a random degree-t polynomial with $f(0) = s$. Then, each party in $\mathcal{L}_{c_1}$ (privately) reconstructs towards party $P_i^{c'}$ the value $s_i = f(i)$. Notice that layer $\mathcal{L}_{c'}$ also holds sharings of all values $f(j)$ for $j \in [1, n]$ thanks to linearity of our VSS. This version of VSS can also be similarly duplicated.

Circuit Evaluation. Input parties use 2-level VSS to distribute their inputs towards a future layer. For each computation gate we maintain the invariant that layer $\mathcal{L}_c$ holds sharings (resulting from the 2-level VSS) of the input wire values x and y, and some future layer $\mathcal{L}_{c'}$ for $c' \geq c+6$ holds a sharing of the output wire value z. Addition gates are processed locally, exploiting the linearity of 2-level-VSS. Multiplication gates are processed by adapting a well-known protocol of Cramer et al. [14]: each party $P_i^c \in \mathcal{L}_c$ holds (as part of their 2-level VSS states to x and y) Shamir shares x_i and y_i of each value, and computes a 2-level VSS for x_i, y_i (but using the already known sharing to the constant coefficient of the used polynomial) and a fresh 2-level VSS for $z_i = x_i \cdot y_i$ towards a future layer. Finally, each party carries out a distributed *multiplication proof* (adapted from [14]) to prove that indeed $z_i = x_i \cdot y_i$: if this proof fails, parties jointly reconstruct (and adopt a standard sharing of) x_i and y_i to continue the computation.

Theorem 16. *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 $t < n/2$, there is an $(n, t, 4+6D)$-layered MPC protocol for f assuming non-interactive linearly-homomorphic equivocal commitments. The communication complexity is $M \cdot O(n^9)$ field elements over the point-to-point channels and $M \cdot O(n^5)$ field elements + $M \cdot O(n^{10} \cdot \lambda)$ bits over the broadcast channels, where λ is the security parameter.*

Acknowledgement. We thank the anonymous reviewers for helpful comments. B. David was supported by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B. A. Konring was supported by IRFD (TrA^{2}C) and by the Otto Mønsted Foundation in a joint program with Innovation Center Denmark - Israel. Y. Ishai, E. Kushilevitz, and V.

Narayanan were supported by ISF grant 2774/20 and BSF grant 2018393. Y. Ishai and V. Narayanan were also supported by ERC Project NTSC (742754).

References

1. Acharya, A., Hazay, C., Kolesnikov, V., Prabhakaran, M.: SCALES - MPC with small clients and larger ephemeral servers. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022, Part II. LNCS, vol. 13748, pp. 502–531. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-22365-5_18
2. Almansa, J.F., Damgård, I., Nielsen, J.B.: Simplified threshold RSA with adaptive and proactive security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 593–611. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_35
3. Baron, J., Defrawy, K.E., Lampkins, J., Ostrovsky, R.: Communication-optimal proactive secret sharing for dynamic groups. In: Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M. (eds.) ACNS 2015. LNCS, vol. 9092, pp. 23–41. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-28166-7_2
4. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC, pp. 1–10. ACM Press, May 1988
5. Benhamouda, F., et al.: Can a public blockchain keep a secret? In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12550, pp. 260–290. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64375-1_10
6. Blum, E., Katz, J., Liu-Zhang, C.-D., Loss, J.: Asynchronous byzantine agreement with subquadratic communication. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12550, pp. 353–380. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64375-1_13
7. Cachin, C., Kursawe, K., Lysyanskaya, A., Strobl, R.: Asynchronous verifiable secret sharing and proactive cryptosystems. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 88–97 (2002)
8. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. CRYPTOLOGY **13**(1), 143–202 (2000)
9. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press, October 2001
10. Canetti, R., Damgard, I., Dziembowski, S., Ishai, Y., Malkin, T.: Adaptive versus non-adaptive security of multi-party protocols. J. Cryptology **17**, 153–207 (2004)
11. Canetti, R., Herzberg, A.: Maintaining security in the presence of transient faults. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 425–438. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48658-5_38
12. Chen, J., Micali, S.: Algorand: a secure and efficient distributed ledger. Theor. Comput. Sci. **777**, 155–183 (2019)
13. Choudhuri, A.R., Goel, A., Green, M., Jain, A., Kaptchuk, G.: Fluid MPC: secure multiparty computation with dynamic participants. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 94–123. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_4
14. Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., Rabin, T.: Efficient multi-party computations secure against an adaptive adversary. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 311–326. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_22

15. Cramer, R., Damgård, I., Maurer, U.: General secure multi-party computation from any linear secret-sharing scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_22
16. Cramer, R., Damgård, I., Nielsen, J.B.: Secure Multiparty Computation and Secret Sharing. Cambridge University Press, Cambridge (2015). https://www.cambridge.org/de/academic/subjects/computer-science/cryptography-cryptology-and-coding/secure-multiparty-computation-and-secret-sharing?format=HB&isbn=9781107043053
17. Damgård, I., Ishai, Y.: Constant-round multiparty computation using a black-box pseudorandom generator. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 378–394. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_23
18. Damgård, I., Escudero, D., Polychroniadou, A.: Phoenix: secure computation in an unstable network with dropouts and comebacks. Cryptology ePrint Archive, Paper 2021/1376 (2021), https://eprint.iacr.org/2021/1376
19. David, B., Konring, A., Ishai, Y., Kushilevitz, E., Narayanan, V.: Perfect MPC over layered graphs. Cryptology ePrint Archive, Paper 2023/330 (2023). https://eprint.iacr.org/2023/330
20. Deligios, G., Goel, A., Liu-Zhang, C.D.: Maximally-fluid MPC with guaranteed output delivery. Cryptology ePrint Archive, Paper 2023/415 (2023). https://eprint.iacr.org/2023/415
21. Desmedt, Y., Jajodia, S.: Redistributing secret shares to new access structures and its applications. Technical Report, Citeseer (1997)
22. Eldefrawy, K., Lepoint, T., Leroux, A.: Communication-efficient proactive secret sharing for dynamic groups with dishonest majorities. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 2020. LNCS, vol. 12146, pp. 3–23. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57808-4_1
23. Feldman, P., Micali, S.: Byzantine agreement in constant expected time (and trusting no one). In: 26th FOCS, pp. 267–276. IEEE Computer Society Press, October 1985
24. Fitzi, M., Garay, J.A.: Efficient player-optimal protocols for strong and differential consensus. In: Borowsky, E., Rajsbaum, S. (eds.) 22nd ACM PODC. pp. 211–220. ACM, July 2003
25. Fitzi, M., Liu-Zhang, C.D., Loss, J.: A new way to achieve round-efficient byzantine agreement. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing, pp. 355–362 (2021)
26. Garay, J.A.: Reaching (and maintaining) agreement in the presence of mobile faults. In: Tel, G., Vitányi, P. (eds.) WDAG 1994. LNCS, vol. 857, pp. 253–264. Springer, Heidelberg (1994). https://doi.org/10.1007/BFb0020438
27. Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: The round complexity of verifiable secret sharing and secure multicast. In: 33rd ACM STOC, pp. 580–589. ACM Press, July 2001
28. Gentry, C., et al.: YOSO: You Only Speak Once. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 64–93. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_3
29. Ghinea, D., Goyal, V., Liu-Zhang, C.D.: Round-optimal byzantine agreement. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part I. LNCS, vol. 13275, pp. 96–119. Springer, Heidelberg (May / Jun (2022). https://doi.org/10.1007/978-3-031-06944-4_4

30. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles, pp. 51–68 (2017)
31. Goldreich, O.: Foundations of Cryptography: volume 2, Basic Applications. Cambridge University Press, Cambridge (2009)
32. Goyal, V., Kothapalli, A., Masserova, E., Parno, B., Song, Y.: Storing and retrieving secrets on a blockchain. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022, Part I. LNCS, vol. 13177, pp. 252–282. Springer, Heidelberg (Mar (2022)
33. Halevi, S., Ishai, Y., Jain, A., Kushilevitz, E., Rabin, T.: Secure multiparty computation with general interaction patterns. In: Sudan, M. (ed.) ITCS 2016, pp. 157–168. ACM, January 2016
34. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: how to cope with perpetual leakage. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 339–352. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-44750-4_27
35. Hirt, M., Maurer, U.M.: Player simulation and general adversary structures in perfect multiparty computation. J. Cryptol. **13**(1), 31–60 (2000), https://doi.org/10.1007/s001459910003
36. Katz, J., Koo, C.-Y.: On expected constant-round protocols for byzantine agreement. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 445–462. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_27
37. Kushilevitz, E., Lindell, Y., Rabin, T.: Information-theoretically secure protocols and security under composition. SIAM J. Comput. **39**(5), 2090–2112 (2010). https://doi.org/10.1137/090755886
38. Maram, S.K.D., et al.: CHURP: dynamic-committee proactive secret sharing. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 2369–2386. ACM Press, November 2019
39. Maurer, U.: Secure multi-party computation made simple. Discrete Appl. Math. **154**(2), 370–381 (2006)
40. Micali, S.: Very simple and efficient byzantine agreement. In: Papadimitriou, C.H. (ed.) ITCS 2017, vol. 4266, pp. 6:1–6:1. LIPIcs, 67, January 2017
41. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: Logrippo, L. (ed.) 10th ACM PODC, pp. 51–59. ACM, August 1991
42. Pass, R., Shi, E.: The sleepy model of consensus. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 380–409. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_14
43. Schultz, D., Liskov, B., Liskov, M.: Mpss: mobile proactive secret sharing. ACM Trans. Inf. Syst. Secur. (TISSEC) **13**(4), 1–32 (2010)
44. Wong, T.M., Wang, C., Wing, J.M.: Verifiable secret redistribution for archive systems. In: First International IEEE Security in Storage Workshop, 2002. Proceedings, pp. 94–105. IEEE (2002)