PUFF: Maximally Proactive Security for Free in Perfectly Secure MPC with Guaranteed Output Delivery

Jiarui Li[1,3], Mengzhen Zou[1], Guidong Li[1], Guoyan Zhang[2,3,4,5✉], and Chen Qian[1,3,5✉]

[1] School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, 266237, China

[2] School of Cryptologic Science and Engineering, Shandong University, Jinan, Shandong, 250101, China

[3] State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Jinan, 250100, China

[4] Shandong Key Laboratory of Artificial Intelligence Security, Shandong University, Jinan , Shandong, 250100, China

[5] Key Laboratory of Cryptologic Technology and Information Security of Ministry of Education, Shandong University, Jinan, Shandong, 250100, China

Abstract. Achieving proactive security in perfectly-secure Multi-Party Computation (MPC) with guaranteed output delivery is a significant challenge, primarily because traditional protocols require all participants to be continuously online, rendering them impractical for many applications. The recently proposed layered MPC model [DDG+23] addresses this by allowing parties to be offline for extended periods. However, existing protocols for this model incur substantial overhead compared to their counterparts in the standard static setting.

This work introduces a unified framework and essential building blocks for constructing protocols in the layered model, instantiable with both Shamir and CNF secret sharing. Using this framework, we develop highly efficient protocols for Verifiable Secret Sharing (VSS) and secure multiplication for proactive security.

Applying our framework, we construct layered MPC protocols that drastically reduce the communication complexity and the number of layers required to evaluate an arithmetic circuit of depth D. Specifically, our Shamir-based MPC achieves $O(\mathsf{n}^6)$ per-gate communication with a total layer depth of $D + 13$, representing a significant improvement over the $O(\mathsf{n}^9)$ complexity and $10D + 8$ depth of [DDG+23].

Keywords. Multi-Party Computation, Verifiable Secret Sharing, Secure Multiplication, Maximally Proactive Security, Dimension Transformation

1 Introduction

Secure multi-party computation (MPC) [Yao86,GMW87,BGW88,CCD88] allows a set of parties to jointly compute a function on their private inputs without revealing them. Classical MPC protocols, however, typically assume a static set of parties that must remain online for the entire duration of the computation. This assumption is impractical for long-running applications or settings with intermittent participation, such as federated learning and blockchain.

1.1 Synchronous Verifiable Secret Sharing with Perfect Security

Verifiable secret sharing (VSS) is a fundamental building block for secure multi-party computation (MPC). A long line of research has established its theoretical limits, with Dolev et al. [DDWY93] proving that any perfectly secure VSS protocol requires an honest majority of parties, i.e., $\mathsf{n} > 3\mathsf{t}$. The first protocol to meet this bound was proposed by Ben-Or, Goldwasser, and Wigderson (BGW) [BGW88]. It employed bivariate polynomials to enable cross-checking of shares, but its round complexity was seven.

Subsequent work focused on reducing this round complexity. Gennaro et al. [GIKR01] presented two VSS protocols for the $\mathsf{n} > 3\mathsf{t}$ setting: a four-round Shamir-based scheme with $O(\mathsf{n}^2)$ communication and a three-round scheme based on additive secret sharing with exponential communication. They also proved that three rounds are optimal for this setting. The challenge then shifted to designing a round-optimal Shamir-based VSS. Katz and Koo [KKK08] achieved this by introducing weak VSS (WSS) as a primitive, resulting in a three-round protocol with $O(\mathsf{n}^3)$ communication. More recently, Applebaum et al. [AKP20] presented an alternative three-round, $O(\mathsf{n}^3)$ Shamir-based VSS using a symmetric bivariate polynomial framework.

All the Shamir-based methods above follow the BGW-style complain-and-resolve paradigm: they cross-check shares via a degree-(t,t) bivariate polynomial and, when disputes arise, require the dealer to speak again for resolution. This pattern is ill-suited for dynamic-committee settings as it requires the dealer to remain online.

1.2 Proactive Security and Fluid MPC

The proactive security model, introduced by Ostrovsky and Yung [OY91], considers a mobile adversary that corrupts different parties over time. To counter this, parties periodically execute a refresh protocol to heal from compromises. This paradigm was recently extended to settings with dynamic participation, leading to two main models: YOSO (You-Only-Speak-Once) [GHK+21] and Fluid MPC [CGG+21]. In these models, the computation is carried out by a sequence of committees, where each round may involve a different set of n parties.

In the YOSO model, parties rely on ideal target-anonymous channels to communicate with future, randomly selected committee members, and the adversary is restricted to probabilistic corruptions. In contrast, the Fluid MPC model

assumes a stronger, worst-case adversary that can corrupt up to a threshold of parties within any given committee, without requiring specialized communication channels.

Much of the research in the fluid model has focused on constructions security with abort [CGG+21,RS22], where protocols with linear communication complexity have been developed [BEP23,BEP25]. However, security with abort is insufficient for applications like electronic voting and auctions, which require guaranteed output delivery (G.O.D.). Achieving G.O.D. in a fluid setting is particularly challenging because traditional protocols rely on a static set of parties to resolve disputes. To date, the only works to achieve G.O.D. in the fluid model [DDG+23,DKLN24] employ a "Future Messaging" technique, which commits messages required for potential re-broadcasts by the Dealer (or other parties) to future committees. While this effectively adapts the complain-and-resolve to the proactive security model, they do so at the cost of significantly higher complexity compared to their counterparts in the static setting. These protocols operate in the *layered model*, which formalizes the strongest adversarial conditions of fluid MPC and maximal proactivity.

Adopting this layered model, we investigate the feasibility of efficient MPC under the strongest security guarantees: perfect security with guaranteed output delivery. This leads to the central question of our work:

How to build efficient MPC in layered model with perfect security and G.O.D?

1.3 Our Contributions

We answer the above question affirmatively by giving a generic framework to build efficient MPC protocols in the layered model with perfect security and G.O.D. Our framework supports both Shamir and CNF secret sharing schemes. Using our unified framework, our construction of verifiable secret sharing in the layered model is more efficient than the state-of-the-art VSS in the layered model.

We summarize our main contribution as follows:

- A unified view of secret sharing and the introduction of dimension transformation techniques.
- A more efficient VSS in the layered model avoiding the incompatibility between the complaint-and-resolve paradigm and the layered model.
- More efficient offline multiplication and MPC for arithmetic circuits in the layered model.

A unified view of secret sharing and dimension transformation techniques. We present a unified framework that generalizes both Shamir and CNF secret sharing schemes as specific instances of a high-dimensional robust secret sharing scheme. This unified perspective enables a modular protocol design, allowing either scheme to be instantiated as needed. The framework achieves maximally proactive security in the layered model using five rounds generically (with Shamir sharing) or four rounds (with CNF sharing).

We also introduce *dimension transformation*, a technique for converting a k-dimensional sharing into a k'-dimensional one. These techniques are essential for achieving maximally proactive secure MPC in the layered model, as they effectively mitigate cross-layer corruption attacks by increasing the dimension of the secret sharing scheme.

A more efficient VSS in the layered model via avoiding the complain-and-resolve paradigm. By applying this framework, we derive a significantly more efficient verifiable secret sharing protocol in the layered model. Specifically, our Shamir-based VSS achieves five rounds with communication complexity of $O(\mathsf{n}^4 \log |\mathbb{F}| + \mathcal{BC}(\mathsf{n}^4 \log |\mathbb{F}|))$, improving upon the previous best-known result of six rounds with $O(\mathsf{n}^6 \log |\mathbb{F}| + \mathcal{BC}(\mathsf{n}^4 \log |\mathbb{F}|))$ in the same maximally-proactive and perfectly secure model. This efficiency gain stems from our VSS construction based on dimension transformation, which avoids the traditional complain-and-resolve paradigm. Consequently, our design eliminates the need for expensive primitives, such as Future Messages, which are typically required to forward secondary broadcast messages from the dealer and participants to subsequent layers in the event of disputes. Here, $\mathcal{BC}$ denotes communication over a broadcast channel. We provide a concrete comparison with other existing VSS protocols in Table 1.

Moreover, when we instantiate our generic VSS protocol with the CNF-based robust secret sharing scheme, we achieve a four-round CNF-based VSS protocol in the layered model, which is one round fewer than the Shamir-based counterpart. This follows from the fact that in the CNF-based VSS protocol, each party can locally identify inconsistent senders in the previous layer using a majority vote. The details of both VSS protocols are presented in Tab. 4 and Tab. 17, respectively.

Shamir-based VSS Scheme	**Round**	**Comm.**
[GIKR01]	4	$O(\mathsf{n}^2\log\|\mathbb{F}\| + \mathcal{BC}(\mathsf{n}^2\log\|\mathbb{F}\|))$
[AKP20]	3	$O(\mathsf{n}^3\log\|\mathbb{F}\| + \mathcal{BC}(\mathsf{n}^3\log\|\mathbb{F}\|))$
[DDG+23] (Layered Setting)	6	$O(\mathsf{n}^6\log\|\mathbb{F}\| + \mathcal{BC}(\mathsf{n}^4\log\|\mathbb{F}\|))$
Our work (Layered Setting)	5	$O(\mathsf{n}^4\log\|\mathbb{F}\| + \mathcal{BC}(\mathsf{n}^4\log\|\mathbb{F}\|))$
CNF-based VSS[6] Scheme	**Round**	**Comm.**
[GIKR01]	3	$O(\mathsf{n}^2 \cdot \binom{\mathsf{n}}{\mathsf{t}} \log\|\mathbb{G}\| + \mathcal{BC}(\mathsf{n}^2 \cdot \binom{\mathsf{n}}{\mathsf{t}} \log\|\mathbb{G}\|))$
[DDG+23] (Layered Setting)	5	$O(\mathsf{n}^2 \cdot \binom{\mathsf{n}}{\mathsf{t}}^3 \log\|\mathbb{R}\| + \mathcal{BC}(\mathsf{n}^2 \cdot \binom{\mathsf{n}}{\mathsf{t}}^3 \log\|\mathbb{R}\|))$
Our work (Layered Setting)	4	$O(\mathsf{n}^4 \cdot \binom{\mathsf{n}}{\mathsf{t}}^4 \log\|\mathbb{G}\| + \mathcal{BC}(\mathsf{n}^4 \cdot \binom{\mathsf{n}}{\mathsf{t}}^4 \log\|\mathbb{G}\|))$

Table 1. Round and communication complexity comparison for perfectly-secure VSS ($\mathsf{n} > 3\mathsf{t}$), with $\mathcal{BC}$ denoting communication over the broadcast channel.

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

More efficient multiplication Our framework also leads to more efficient protocols for multiplication. Compared to the previous best protocols in the layered model [DDG+23], our Shamir-based multiplication protocol reduces the round complexity from 10 to 8 and the communication complexity from $O(\mathsf{n}^9)$ to $O(\mathsf{n}^6)$, thereby matching BGW in the standard model. We give a direct comparison in Table 2.

Shamir Multiplication	Round	Comm.
[BGW88] [7]	6 ($\mathsf{R_{VSS}}+3$)	$O(\mathsf{n}^2 \cdot \mathsf{Comm_{VSS}} + \mathsf{n}^5)$
[DDG+23] (Layered Setting)	10 ($\mathsf{R_{VSS}}+4$)	$O(\mathsf{n}^9)$
Our work (Layered Setting)	8 ($\mathsf{R_{VSS}}+3$)	$O(\mathsf{n}^6)$

Table 2. Comparison of the round and communication complexity for Shamir Multiplication($\mathsf{n} > 3\mathsf{t}$). $\mathsf{R_{VSS}}$ and $\mathsf{Comm_{VSS}}$ denotes round and communication complexity of VSS used in the multiplication.

More efficient MPC for arithmetic circuits in the layered model. Building on our improved multiplication protocol, we construct Shamir-based and CNF-based MPC protocols in the layered model. Our Shamir-based MPC protocol evaluates an arithmetic circuit of depth D using only $D + 13$ layers, a significant improvement over the prior $10D + 8$ layers [DDG+23]. Similarly, our CNF-based MPC protocol requires only $D + 11$ layers, compared to the previous $7D + 6$ layers. We summarize the round complexity comparison in Tab. 3. In addition, we extend layered MPC, originally defined only for layered arithmetic circuits[8], to support arbitrary arithmetic circuits.

MPC protocol	Round(Shamir-based)	Round(CNF-based)
[BGW88]	$D + O(1)$	–
[DDG+23] (Layered Setting)	$10D + 8$	$7D + 6$
Our work (Layered Setting)	$D + 13$	$D + 10$

Table 3. Round complexity comparison for MPC Protocol(evaluating a depth-D arithmetic circuit). For BGW, we adopt the standard preprocessing with Beaver triples: multiplication triples are generated offline. We treat the rounds of offline as $O(1)$.

[7] The round complexity and communication complexity of the multiplication protocol are not provided in [BGW88]. In [AL17], Section 6 presents the details and complexity of the protocol. Additionally, We choose 3-round VSS to obtain optimal round complexity.

[8] *Layered arithmetic circuits* impose a layering on a standard arithmetic circuit: the gates are partitioned into $L_0, \ldots, L_D$ (with inputs in L_0), and every gate $g \in L_i$ takes its inputs only from L_{i-1}. Hence edges go strictly from L_{i-1} to L_i; no skip-layer dependencies are allowed.

2 Technical Overview

We begin by outlining the main technical challenges in adapting existing secure multi-party computation (MPC) protocols to the layered setting, particularly concerning the preservation of security and efficiency.

2.1 A Unified View of Secret Sharing

Secret sharing is a cornerstone of MPC. The literature is dominated by two primary schemes: the Shamir secret sharing [Sha79], favored for constructing communication-efficient protocols, and the CNF secret sharing [ISN89], often used to achieve minimal round complexity. Although some techniques are transferable, these schemes are typically treated as distinct paradigms.

In this work, we introduce a unified framework that generalizes both. We show that Shamir and CNF sharing are specific instances of a high-dimensional robust secret sharing scheme. This unified perspective enables a modular design for our protocols, allowing either scheme to be plugged in as needed.

Furthermore, we extend this framework to support *dimension transformation*, a mechanism for converting a k-dimensional sharing into a k'-dimensional one. The construction is recursive, with standard 1-dimensional Shamir or CNF sharing as the base case. For instance, a k-dimensional Shamir sharing of a secret s is created by sampling a random polynomial $\mathsf{F}(X_k, \ldots, X_1)$ of degree at most t in each variable, such that $\mathsf{F}(0, \ldots, 0) = s$.

2.2 Dimension Transformation and Proactive Security

A key challenge in achieving maximally proactive security is that the adversary could corrupt different parties in different layers, then create a cross-layer *corruption cohesion* attack. Our first observation is that the corruption cohesion attack can be mitigated by increasing the dimension of the secret sharing scheme.

Example Consider a protocol spanning two layers using a 2-dimensional (bivariate) Shamir secret sharing scheme in an (n, t)-layered MPC setting. Let a secret s be shared via a random bivariate polynomial $\mathsf{F}(X_1, X_2)$ of degree t in each variable, where $s = \mathsf{F}(0, 0)$. Each party P_i^1 in the first layer, $\mathcal{L}_1$, holds the univariate polynomial $\mathsf{F}(i, X_2)$ as its share. To transfer the sharing to the next layer, P_i^1 sends the point evaluation $\mathsf{F}(i, j)$ to party P_j^2 in $\mathcal{L}_2$. After receiving points from all parties in $\mathcal{L}_1$, each party P_j^2 in $\mathcal{L}_2$ can reconstruct its share, the univariate polynomial $\mathsf{F}(X_1, j)$, by interpolating the points it received.

Now, consider an adversary who corrupts t parties in each layer. In $\mathcal{L}_1$, the adversary learns t polynomials of the form $\mathsf{F}(i, X_2)$. In $\mathcal{L}_2$, the adversary learns another t polynomials of the form $\mathsf{F}(X_1, j)$. This corresponds to learning the values of F on a $\mathsf{t} \times \mathsf{n}$ grid and an $\mathsf{n} \times \mathsf{t}$ grid of points. The union of these points forms a $(\mathsf{t} \times \mathsf{n}) \cup (\mathsf{n} \times \mathsf{t})$ grid. Crucially, the largest square subgrid of points known to the adversary is of size $\mathsf{t} \times \mathsf{t}$. Since reconstructing the degree-(t, t) polynomial

$\mathsf{F}(X_1, X_2)$ requires knowledge of its values on at least a $(\mathsf{t}+1) \times (\mathsf{t}+1)$ grid of points, the $\mathsf{t} \times \mathsf{t}$ points are insufficient to learn any information about the secret $s = \mathsf{F}(0, 0)$.

This example illustrates a general principle. A k-dimensional secret sharing can be visualized as a k-dimensional hypercube of values. Passing shares from one layer to the next corresponds to rotating then projecting this hypercube along one of its dimensions. An adversary corrupting t parties in each of the k layers learns t hyperplanes of values in each of the k dimensions. This information is insufficient to reconstruct the secret, as that would require learning a $(\mathsf{t}+1) \times \cdots \times (\mathsf{t}+1)$ sub-hypercube of values. Thus, by matching the dimension of the secret sharing to the number of layers, we can prevent the adversary from combining information across layers to compromise the secret.

Layered MPC The above insight leads to a straightforward approach for achieving maximally proactive security in layered MPC: employ a k-dimensional secret sharing scheme for a protocol spanning k layers. Each layer uses one dimension of the sharing, and shares are passed to the next layer by projecting along that dimension. This ensures that even if the adversary corrupts t parties in each layer, they cannot gather enough information to reconstruct the secret. This perspective not only simplifies the design of secure protocols in the layered setting but also provides a clear framework for analyzing their security.

2.3 Dimension Transformation and Complain-Resolve

Another challenge in the layered setting is handling complaints and resolutions when parties are not static but change across layers. In traditional MPC protocols, if a party suspects misbehavior, it can raise a complaint, and the same set of parties can engage in a resolution process. However, in the layered model, the parties involved in the complaint is not present in the future layer, making it unclear who should handle the resolution.

To address this, we leverage our dimension transformation technique. Our second observation is that we do not necessarily need to fold the dimension after handing the hypercube to the next layer. Instead, we can maintain the higher-dimensional structure and allow parties in the future layer to access the relevant information from the previous layer's hypercube. This approach ensures that even if the parties involved in the complaint are not present in the future layer, the necessary information for resolution is still accessible for honest parties.

2.4 Towards the Most Efficient Verifiable Secret Sharing

Combine the above two observations, we can now outline our approach to constructing an efficient verifiable secret sharing (VSS) protocol in the layered setting. We give the intuition behind our 5 layer VSS protocol with the Shamir-based secret sharing and 4 layer VSS one with the CNF-based secret sharing.

To launch the protocol, the dealer first shares its secret using a 4-dimensional secret sharing scheme. Visually, this can be represented as a 4-dimensional cube of values. Then, the dealer distributes the 3-dimensional shares (hyperplanes of the cube) to parties in the first layer, $\mathcal{L}_1$. Each party in $\mathcal{L}_1$ receives a 3-dimensional slice of the hypercube, corresponding to fixing one dimension of the 4-dimensional polynomial.

A key property of VSS is verifiability: the ability for honest parties to detect a faulty dealer. In our protocol, a dealer is deemed faulty if the distributed 3-dimensional cube of shares does not contain any good sub-cube. The error-correction property of the underlying robust secret sharing scheme allows for the recovery of the polynomial from such a sub-cube. To ensure verifiability, our protocol satisfies the following three properties:

- **Correctness:** If the dealer is honest, at least one such good sub-cube exists.
- **Soundness:** If no such good sub-cube exists, all honest parties will detect the dealer's fault.
- **Uniqueness:** If such a good sub-cube exists, all honest parties can reconstruct the same unique secret from it.

The correctness property is straightforward. An honest dealer samples a random 3-dimensional polynomial of degree $\mathtt{t}$ in each variable. The resulting cube of shares naturally contains many valid $(\mathtt{t}+1)^3$ sub-cubes. Noticing that the hypercube is only defined by the slices received by honest parties in $\mathcal{L}_1$, the existence of a good sub-cube is guaranteed.

For soundness, the dealer distributes shares such that a good 3-dimensional $(2\mathtt{t}+1)^3$ sub-cube exists. Parties in each layer check for this and publicly broadcast any detected inconsistencies. However, such checks may produce false positives: even if all public checks pass, the dealer may collude with the adversary to introduce inconsistencies into the shares received by parties in the final layer $\mathcal{L}_5$, since among the $2\mathtt{t}+1$ parties claiming inconsistency in each layer, up to $\mathtt{t}$ may be malicious. Fortunately, we can rely on the existence of $(\mathtt{t}+1)^2$ "inconsistency-free" sub-cubes, which contain shares only from honest parties and are thus guaranteed to be consistent. Honest parties use these sub-cubes to publicly recover any inconsistent shares held by honest parties outside of the good sub-cube. This public recovery is verifiable by all parties in $\mathcal{L}_5$. If the dealer is faulty and the corrupted parties refuse to publicly recover inconsistent shares, all honest parties in $\mathcal{L}_5$ will detect the fault by verifying consistency. Conversely, if the dealer is honest, all shares from honest parties in $\mathcal{L}_5$ will be consistent, and the dealer is deemed honest.

2.5 Efficient Multiplication and MPC

Building on our efficient VSS protocol and dimension transformation techniques, we construct a multiplication protocol for the layered setting. Our approach adapts the classic BGW multiplication method [BGW88], including its degree-reduction and resharing steps, to the layered model.

Integrating these components, we construct our final MPC protocol. As summarized in Table 3, our protocol achieves significant improvements in both round and communication complexity over prior work [DDG+23]. Notably, our Shamir-based MPC protocol matches the round complexity of the classic BGW protocol in the standard model. This demonstrates that our techniques can bridge the gap between the static and dynamic committee settings without incurring a round-complexity overhead.

3 Organization

This paper is organized as follows. Section 4 introduces the necessary preliminaries, including layered MPC, robust secret sharing, and verifiable secret sharing. Section 5 presents our multi-dimensional robust secret sharing scheme which generalizes both Shamir and CNF secret sharing, a fundamental building block for our constructions. In Section 6, we leverage this scheme to build several cross-layer messaging functionalities, which are key to achieving proactive security in our protocols. Section 7 details our main technical contribution: a parallel verifiable secret sharing protocol with low communication and round complexity. Section 8 then presents our offline layered multiplication protocol. Finally, Section 9 combines these components to construct a complete MPC protocol for general functionalities that achieves maximal proactive security.

4 Preliminaries

4.1 Notations

We denote by $\mathbb{F}$ a finite field. For a positive integer n, we denote by $[\mathsf{n}]$ the set $\{1, 2, \ldots, \mathsf{n}\}$. We use bold uppercase letters (e.g., $\mathbf{A}$) to denote matrices, bold lowercase letters (e.g., $\mathbf{a}$) to denote vectors, and regular lowercase letters (e.g., a) to denote scalars. For a matrix $\mathbf{A}$, we use $\mathbf{A}[i, j]$ to denote the entry in the i-th row and j-th column of $\mathbf{A}$, and $\mathbf{A}[i, \cdot]$ to denote the i-th row of $\mathbf{A}$. For a vector $\mathbf{a}$, we use $\mathbf{a}[i]$ to denote the i-th entry of $\mathbf{a}$. We use calligraphic uppercase letters (e.g., $\mathcal{A}$) to denote sets. For a set $\mathcal{A}$, we use $|\mathcal{A}|$ to denote its cardinality. We use bold script letters (e.g., ss) to denote vectors of shares.

4.2 Layered MPC

We begin by defining layered MPC [DDG+23] in the UC framework. After this, we recall the definitions of robust secret sharing and verifiable secret sharing, which serve as the core building blocks for constructing MPC protocols.

Maximal Proactive Security The layered MPC model of [DDG+23] is a specialization of standard MPC with general adversary structures, characterized by: (i) an interaction pattern defined by a layered graph; and (ii) an adversary

that may corrupt at most t parties per layer. In [DDG+23], it was shown that, assuming secure erasures, secure layered MPC implies secure maximally proactive MPC. We do not repeat the proof; instead, we recall the formal definition of layered MPC as follows:

Definition 1 (Layered MPC). *Let* $\mathsf{n}, \mathsf{t}, \mathsf{d}$ *be positive integers. An* $(\mathsf{n}, \mathsf{t}, \mathsf{d})$*-layered protocol is a synchronous protocol* $\mathbf{\Pi}$ *over secure point-to-point channels and a broadcast channel, with the following special features.*

Parties: *There are* $\mathsf{N} = \mathsf{n}(\mathsf{d}+1)$ *parties, partitioned into* $\mathsf{d}+1$ *layers* $\mathcal{L}_i$ *for* $0 \leq i \leq \mathsf{d}$*, with* $|\mathcal{L}_i| = \mathsf{n}$*. Parties in the last layer* $\mathcal{L}_\mathsf{d}$ *are referred to as output clients. Parties in the first layer* $\mathcal{L}_0$ *are referred to as input clients. Parties in intermediate layers* $\mathcal{L}_i$ *for* $1 \leq i < \mathsf{d}$ *are referred to as computation parties.*

Interaction Pattern: *The execution of the protocol consists of* d *rounds. In round* $i \in \{1, \ldots, \mathsf{d}\}$*, parties in* $\mathcal{L}_{i-1}$ *may send messages to parties in* $\mathcal{L}_i$ *over secure point-to-point channels. Each party in* $\mathcal{L}_{i-1}$ *may also broadcast to all parties in* $\mathcal{L}_i$*.*

Functionalities: *We consider functionalities* $\mathcal{F}$ *that take inputs from the input clients and deliver outputs to the output clients.*

Adversaries: *We allow an active, rushing, adaptive adversary that may corrupt any number of input and output clients, in addition, up to* t *parties in each intermediate layer* $\mathcal{L}_i$ *for* $0 < i < \mathsf{d}$*.*

We say that a protocol $\mathbf{\Pi}$ *is a layered MPC protocol for* $\mathcal{F}$ *if it realizes* $\mathcal{F}$ *in the standard sense of (stand-alone) secure MPC with general adversary structures [Can00]. We consider the case of perfect security with guaranteed output delivery where* $\mathsf{t} < \mathsf{n}/3$*.*

Universal Composability Following [DDG+23], we develop our protocol for general functionalities in the synchronous, stand-alone model. We prove its perfect security via a straight-line black-box simulator. To extend this result to the Universal Composability (UC) framework, we can adopt one of two standard techniques, as in [DDG+23]:

- Insert dummy layers between every two layers of the original protocol. This ensures synchrony and allows us to apply the result of [KLR06] to establish UC security.
- Broadcast the messages for the initial VSS subprotocols to all parties. Then, the protocol is secure under the Micali-Rogaway definition of security [MR92]. Subsequently, applying the result of Dodis and Micali [DM00] lifts the security guarantee to the UC framework.

4.3 Strongly Robust Secret Sharing

Definition 2 (Robust Secret Sharing). *A* (n, t)*-robust secret sharing scheme over message space* $\mathcal{M}$ *over an Abelian group* $\mathbb{G}$ *consists of a tuple of algorithms* $\mathsf{SS} = (\mathsf{Share}, \mathsf{Reconst})$ *with both message and share space in* $\mathcal{M} = \mathbb{G}$*, and the following syntax:*

$\mathsf{Share}(\mathsf{m}) \to [\![\mathsf{m}]\!]$: *Takes a message* $\mathsf{m} \in \mathcal{M}$ *as input, and outputs a vector of shares* $\mathsf{ss} = ([\![\mathsf{m}]\!][1], \ldots, [\![\mathsf{m}]\!][\mathsf{n}]) \in \mathcal{M}^{\mathsf{n}}$.

$\mathsf{Reconst}([\![\mathsf{m}]\!][1], \ldots, [\![\mathsf{m}]\!][\mathsf{n}]) \to \mathsf{m}$: *Takes* n *shares as input, and outputs a message* $\mathsf{m} \in \mathcal{M} \cup \{\perp\}$.

We require the following properties to hold:

Correctness: *For any* $\mathsf{m} \in \mathcal{M}$*, we have*

$$\forall I \subseteq [\mathsf{n}] : |I| \geq \mathsf{t}+1 \implies \Pr[\mathsf{Reconst}(\{[\![\mathsf{m}]\!][i]\}_{i\in I}, \perp, \ldots, \perp) = \mathsf{m} \mid [\![\mathsf{m}]\!] \xleftarrow{\$} \mathsf{Share}(\mathsf{m})] = 1.$$

Robustness: *For any* $\mathsf{m} \in \mathcal{M}$*, let* $[\![m]\!] \xleftarrow{\$} \mathsf{Share}(\mathsf{m})$*, for all vectors of shares* ss*, we have*

$$\exists I \subseteq [\mathsf{n}], \forall i \in I : \mathsf{ss}[i] = [\![m]\!][i] \wedge |I| \geq \mathsf{n}-\mathsf{t} \implies \mathsf{Reconst}(\mathsf{ss}) = \mathsf{m}.$$

Secrecy: *For all* $\mathsf{m}, \mathsf{m}' \in \mathcal{M}$*, for all* $S \subseteq [\mathsf{n}]$ *with* $|S| < \mathsf{t}$*, the following distributions are identical:*

$$\{\{[\![\mathsf{m}]\!][i]\}_{i\in S} : [\![\mathsf{m}]\!] \xleftarrow{\$} \mathsf{Share}(\mathsf{m})\} \equiv \{\{[\![\mathsf{m}']\!][i]\}_{i\in S} : [\![\mathsf{m}']\!] \xleftarrow{\$} \mathsf{Share}(\mathsf{m}')\}.$$

Additively Homomorphic: *For all* $\mathsf{m}_1, \mathsf{m}_2 \in \mathcal{M}$*, and all* $\gamma \in \mathbb{F}$*, we have*

$$\Pr\left[\mathsf{Reconst}([\![\mathsf{m}_1]\!] + \gamma \cdot [\![\mathsf{m}_2]\!]) = \mathsf{m}_1 + \gamma \cdot \mathsf{m}_2 \;\middle|\; \begin{array}{l} [\![\mathsf{m}_1]\!] \xleftarrow{\$} \mathsf{Share}(\mathsf{m}_1) \\ [\![\mathsf{m}_2]\!] \xleftarrow{\$} \mathsf{Share}(\mathsf{m}_2) \end{array}\right] = 1.$$

Strong Robustness: *For any* $\mathsf{m} \in \mathcal{M}$*, let* $[\![m]\!] \xleftarrow{\$} \mathsf{Share}(\mathsf{m})$*, for all vectors of shares* ss *such that* $\#\{i \mid \mathsf{ss}[i] \neq [\![m]\!][i]\} \leq \mathsf{t}$*, we have a deterministic polynomial-time algorithm* ErrorDetect *such that*

$$\mathsf{ErrorDetect}(\mathsf{ss}) = \{i \mid \mathsf{ss}[i] \neq [\![m]\!][i]\}.$$

Moreover, there exists a deterministic polynomial-time algorithm Repair *such that for any* ss *and any* $i \in [\mathsf{n}]$*, we have*

$$\mathsf{Repair}(\mathsf{ss}, i) = [\![m]\!][i].$$

Moreover, we say a vector of shares ss *is* consistent *if* $\mathsf{ErrorDetect}(\mathsf{ss}) = \emptyset$.

4.4 Verifiable Secret Sharing

We adapt the notion of verifiable secret sharing (VSS) to the layered setting. More precisely, since we do not argue the universal composability of the VSS, we define a new ideal functionality in the standalone model for parallel VSS, which consists of l parallel instances of the VSS functionality from $\mathcal{L}_0$ to $\mathcal{L}_d$. The formal definition is provided in Section A, as well as the ideal functionalities of multiplication and layered-MPC.

4.5 Randomness Generation

We define a randomness generation functionality $\mathcal{F}_{\mathsf{RandGen}}$ that allows parties in layer $\mathcal{L}_i$ to jointly generate a shared random value, which is then secret shared among parties in layer $\mathcal{L}_{i+1}$. The formal definition is provided in Section A.

In Shamir-based secret sharing, we use a public Vandermonde matrix to linearly mix the shares of n random values, so generating $\mathsf{n} - \mathsf{t}$ globally shared random values requires only one $\mathcal{F}_{\mathsf{Par\text{-}VSS}}$ instance. We provide the detailed construction for Shamir-based secret sharing and CNF-based secret sharing in Section B.

5 Multi-Dimensional Robust Secret Sharing

We generalize the classical robust secret sharing scheme to the higher dimensional one recursively. More precisely, a k-dimensional robust secret sharing scheme is defined recursively using the $(k-1)$-dimensional robust secret sharing scheme, with 1-dimensional robust secret sharing defined as in Definition 2.

Definition 3 (k-Dimensional Secret Sharing scheme). *A k-dimensional (n, t)-robust secret sharing scheme consists of a tuple of* PPT *algorithms $\mathsf{SS}_k = (\mathsf{Share}, \mathsf{Reconst})$ with message space $\mathcal{M}$, and the similar syntax as in Definition 2. Formally, we have*

- $\mathsf{Share}(\mathsf{m}) \to \mathsf{ss}$: *Takes a message $\mathsf{m} \in \mathcal{M}$ as input and outputs a k-dimensional share ss.*
- $\mathsf{Reconst}(\mathsf{ss}) \to \mathsf{m}$: *Takes a k-dimensional share ss as input and outputs a message $\mathsf{m} \in \mathcal{M} \cup \{\perp\}$.*

We require the correctness, robustness, secrecy and additively homomorphic properties as in Definition 2 to hold for the k-dimensional shares.

5.1 Dimension Projection

A secrecy-preserving projection is a procedure that reduces a k-dimensional robust secret sharing to a vector of $(k-1)$-dimensional ones. The "secrecy-preserving" property ensures that an adversary gains no information about the secret, even when corrupting a threshold number of shares from multiple dimension levels simultaneously. This property, formally defined in Definition 4, is crucial for our layered construction, as it prevents information leakage from adversaries corrupting parties across different layers.

Definition 4 (Secrecy-Preserving Projection). *A (n, t)-robust secret sharing scheme SS_k of dimension k supports secrecy-preserving projection if there exists efficient* deterministic *algorithms $(\mathsf{Proj}, \mathsf{Fold}, \mathsf{ErrorDetect}, \mathsf{Repair}, \mathsf{Rotate})$ with the following syntax:*

- $\mathsf{Proj}(\mathsf{ss}) \to (\mathsf{ss}[1], \ldots, \mathsf{ss}[\mathsf{n}])$: *Takes a k-dimensional share ss of m as input, and outputs a $(k-1)$-dimensional vector of shares $(\mathsf{ss}[1], \ldots, \mathsf{ss}[\mathsf{n}])$.*

$\mathsf{Fold}(\{\mathsf{ss}'[i]\}_{i\in[\mathsf{n}]}) \to \mathsf{ss}$: *Takes a vector of* n *shares of a* k*-dimensional share as input, and outputs a* k*-dimensional share.*

$\mathsf{ErrorDetect}(\mathsf{ss}[1],\ldots,\mathsf{ss}[\mathsf{n}]) \to \mathcal{P}([\mathsf{n}])\cup\perp$: *Takes a vector of* $(k-1)$*-dimensional shares as input and outputs a set of indices of corrupted shares, or* $\perp$ *if the share is inconsistent.*

$\mathsf{Repair}(\mathsf{ss}[1],\ldots,\mathsf{ss}[\mathsf{n}]) \to \mathsf{ss}'$: *Takes a vector of* $(k-1)$*-dimensional shares as input and outputs a repaired* k*-dimensional share* ss'*.*

We also define the rotation operation that "rotate" a 2*-dimensional vector of shares. More precisely, we have*

$\mathsf{Rotate}(\{\mathsf{ss}'[1,j]\}_{j\in[\mathsf{n}]},\ldots,\{\mathsf{ss}'[\mathsf{n},j]\}_{j\in[\mathsf{n}]}) \to (\{\mathsf{ss}[1,j]\}_{j\in[\mathsf{n}]},\ldots,\{\mathsf{ss}[\mathsf{n},j]\}_{j\in[\mathsf{n}]})$: *Takes a* 2*-dimensional vector of* k*-dimensional shares and rotates it, outputting a vector of* k*-dimensional shares, such that* $\mathsf{ss}[i,j]=\mathsf{ss}'[j,i]$ *for all* $i,j\in[\mathsf{n}]$*.*

For simplicity of notation we denote by

$$\mathsf{Proj}(\mathsf{ss}[1],\ldots,\mathsf{ss}[\mathsf{n}]) = (\{\mathsf{ss}[1,j]\}_{j\in[\mathsf{n}]},\ldots,\{\mathsf{ss}[\mathsf{n},j]\}_{j\in[\mathsf{n}]}),$$

the projection over the vector of shares. Note that ss *is* 2*-dimensional vector of* $(k-1)$*-dimensional shares of the form* $\mathsf{ss}[i,j]$*. Similarly we have*

$$\mathsf{Fold}(\mathsf{Fold}(\{\mathsf{ss}[1,i]\}_{i\in[\mathsf{n}]},\ldots,\{\mathsf{ss}[\mathsf{n},i]\}_{i\in[\mathsf{n}]})) = \mathsf{Fold}(\mathsf{ss}'[1],\ldots,\mathsf{ss}'[\mathsf{n}]) = \mathsf{ss}'',$$

for the folding over the 2*-dimensional set of shares. The rotation over multi-dimensional shares is defined as the rotation over the last two dimensions.*

For simplicity of notation, throughout the paper we use Fold *to denote the* robust folding *operator, which first applies* Repair *to the input vector and then folds the repaired vector. We also extend projection to the abort symbol by defining*

$$\mathsf{Proj}(\perp) := (\perp,\ldots,\perp).$$

The projection, folding, error detection, repair, and rotation operations satisfy the following properties:

Correctness: *For any* $\mathsf{m}\in\mathcal{M}$*, we have*

$$\Pr\left[\begin{array}{l|l}\mathsf{SS}_{k-1}.\mathsf{ErrorDetect}(\mathsf{ss})=\emptyset & \mathsf{ss}'\xleftarrow{\$}\mathsf{SS}_k.\mathsf{Share}(\mathsf{m})\\ \wedge\mathsf{SS}_{k-1}.\mathsf{Reconst}(\mathsf{ss})=\mathsf{m} & \mathsf{ss}:=\mathsf{Fold}(\mathsf{Proj}(\mathsf{ss}'))\end{array}\right]=1.$$

Secrecy-Preserving: *For all* $\mathsf{m},\mathsf{m}'\in\mathcal{M}$*, for all* $S_k,\ldots,S_1\subseteq[\mathsf{n}]$ *such that* $|S|_i\le\mathsf{t}$ *for all* $i\in[k]$*, the following distributions are identical:*

$$\left\{\{\mathsf{ss}_k[i]\}_{i\in S_k},\ldots,\{\mathsf{ss}_1[i]\}_{i\in S_1}\middle|\begin{array}{l}\mathsf{ss}_k\xleftarrow{\$}\mathsf{SS}_k.\mathsf{Share}(\mathsf{m})\\ \textbf{for } j\in\{k,\ldots,2\}:\\ \quad\mathsf{ss}_{j-1}:=\mathsf{Proj}(\mathsf{ss}_j)\end{array}\right\}$$

$$\equiv\left\{\{\mathsf{ss}'_k[i]\}_{i\in S_k},\ldots,\{\mathsf{ss}'_1[i]\}_{i\in S_1}\middle|\begin{array}{l}\mathsf{ss}'_k\xleftarrow{\$}\mathsf{SS}_k.\mathsf{Share}(\mathsf{m}')\\ \textbf{for } j\in\{k,\ldots,2\}:\\ \quad\mathsf{ss}'_{j-1}:=\mathsf{Proj}(\mathsf{ss}'_j)\end{array}\right\}.$$

Robust Repair and Error Detection: *For all k-dimensional secret shares* ss'*, we define the repair functionality as*

$$\mathsf{SS}_k.\mathsf{Repair}(\mathsf{ss}') = \begin{cases} \mathsf{ss} & \text{if } \exists!\mathsf{m} \in \mathcal{M}, \exists!\mathsf{ss}, \\ & \quad \mathsf{ss} \in \mathsf{Supp}(\mathsf{SS}_k.\mathsf{Share}(\mathsf{m})) \\ & \quad \wedge \mathsf{Card}\left(\{i \mid \mathsf{Proj}(\mathsf{ss}')[i] \neq \mathsf{Proj}(\mathsf{ss})[i]\}\right) \leq \mathsf{t} \\ \bot & \text{otherwise} \end{cases}.$$

If $\mathsf{Repair}(\mathsf{ss}') = \bot$*, we define* $\mathsf{ErrorDetect}(\mathsf{ss}') = [\mathsf{n}]$*. Otherwise, we define*

$$\mathsf{SS}_k.\mathsf{ErrorDetect}(\mathsf{ss}') = \{i \in [\mathsf{n}] \mid \mathsf{Proj}(\mathsf{ss}')[i] \neq \mathsf{Proj}(\mathsf{Repair}(\mathsf{ss}'))[i]\}.$$

Additionally, we require that for all distinct $\mathsf{m}, \mathsf{m}' \in \mathcal{M}$*, for all secret shares* $\mathsf{ss} \in \mathsf{Supp}(\mathsf{SS}_k.\mathsf{Share}(\mathsf{m}))$ *and* $\mathsf{ss}' \in \mathsf{Supp}(\mathsf{SS}_k.\mathsf{Share}(\mathsf{m}'))$*, we have the Hamming distance between their projections is greater than* t*, i.e.,*

$$d_{\mathsf{H}}(\mathsf{Proj}(\mathsf{ss}), \mathsf{Proj}(\mathsf{ss}')) > \mathsf{t}.$$

Commutativity of Projection and Folding: *Let $k > 1$, for any vector of k-dimensional shares* ss*, we have*

$$\mathsf{Proj}(\mathsf{Fold}(\mathsf{Repair}(\mathsf{ss}))) = \mathsf{Fold}(\mathsf{Proj}(\mathsf{Repair}(\mathsf{ss}))).$$

Rotation Consistency: *Let $k > 1$, for any vector of k-dimensional shares* ss*, we have*

$$\mathsf{ErrorDetect}(\mathsf{ss}) = \emptyset \implies \mathsf{ErrorDetect}(\mathsf{Rotate}(\mathsf{ss})) = \emptyset.$$

Folding Uniqueness: *For any vector of shares* $\mathsf{ss}_1, \mathsf{ss}_2$*, we have*

$$\begin{aligned} \mathsf{Card}\left(\{i \mid \mathsf{ss}_1[i] = \mathsf{ss}_2[i]\}\right) \geq \mathsf{n} - \mathsf{t} & \\ \implies \mathsf{Fold}(\mathsf{ss}_1) = \mathsf{Fold}(\mathsf{ss}_2) & \\ \vee\, \mathsf{Fold}(\mathsf{ss}_1) = \bot \vee \mathsf{Fold}(\mathsf{ss}_2) = \bot & . \end{aligned}$$

The projection operation Proj is a key feature of our multi-dimensional secret sharing scheme. It reduces the dimension of shares while preserving their secrecy. Furthermore, its deterministic nature is essential for the consistency checks in our verifiable secret sharing construction.

Moreover, the projection Proj and the folding Fold generalize the reconstruction procedure. Specifically, reconstructing a secret from a k-dimensional sharing can be viewed as applying a project-rotate-then-fold sequence k times. More specifically, this recursive process can be expressed as:

$$\mathsf{SS}_k.\mathsf{Reconst}([\![\mathsf{m}]\!]) = (\mathsf{Fold} \circ \mathsf{Proj})^k([\![\mathsf{m}]\!]).$$

This perspective is particularly useful for understanding the structure of our multi-dimensional secret sharing scheme.

5.2 Dimension Lifting

As a complementary operation to dimension projection, dimension lifting allows a party to increase the dimension of a $(k-1)$-dimensional robust vector of secret shares to a k-dimensional one. More precisely, given a $(k-1)$-dimensional (n,t)-robust secret sharing of a message m together with r randomly generated $(k-1)$-dimensional (n,t)-robust secret sharing, a party holding the shares can locally compute a k-dimensional (n,t)-robust secret sharing of the same message m without any interaction with other parties. The formal definition of dimension lifting is provided in Definition 5.

Noticing that the parameter r dependents on the robust secret sharing scheme used. For example, for Shamir's robust secret sharing scheme, we have $\mathsf{r}=\mathsf{t}$, while for CNF robust secret sharing scheme, we have $\mathsf{r}=\binom{\mathsf{n}}{\mathsf{t}}$.

Definition 5 (Dimension Lifting). *A (n,t)-robust secret sharing scheme SS_k of dimension k supports dimension lifting if there exists an efficient* randomized *algorithm* Lift *with the following syntax:*

$\mathsf{Lift}(\llbracket\mathsf{m}\rrbracket[i],\llbracket r_1\rrbracket[i],\ldots,\llbracket r_{\mathsf{r}}\rrbracket[i])\to \mathsf{ss}'[i]$: *Takes a $(k-1)$-dimensional share $\llbracket\mathsf{m}\rrbracket[i]$ of m and r $(k-1)$-dimensional shares $\llbracket r_1\rrbracket[i],\ldots,\llbracket r_{\mathsf{r}}\rrbracket[i]$ of random messages as input, and outputs a k-dimensional share vector $\mathsf{ss}'[i]$.*

For simplicity of notation we denote by $\mathsf{Lift}(\llbracket\mathsf{m}\rrbracket,\llbracket r_1\rrbracket,\ldots,\llbracket r_{\mathsf{r}}\rrbracket)=\mathsf{ss}'$ the lifting over the vector of shares. Note that ss' is a k-dimensional vector of shares of the form $\mathsf{ss}'[i]$. Moreover, we require that the lifting operation Lift satisfies the following property:

Correctness: *For any $\mathsf{m}\in\mathcal{M}$, let $\llbracket\mathsf{m}\rrbracket \xleftarrow{\$} \mathsf{SS}_{k-1}.\mathsf{Share}(\mathsf{m})$, and for all $j\in[\mathsf{t}]$, let $\llbracket r_j\rrbracket \xleftarrow{\$} \mathsf{SS}_{k-1}.\mathsf{Share}(r_j)$ for random $r_j\hookleftarrow \mathsf{U}(\mathcal{M})$, we have*

$$\Pr\left[\begin{matrix}\mathsf{SS}_k.\mathsf{Reconst}(\mathsf{ss}')=\mathsf{m}\\ \wedge\mathsf{SS}_k.\mathsf{ErrorDetect}(\mathsf{ss}')=\emptyset\end{matrix}\;\middle|\;\mathsf{ss}':=\mathsf{Lift}(\llbracket\mathsf{m}\rrbracket,\llbracket r_1\rrbracket,\ldots,\llbracket r_{\mathsf{t}}\rrbracket)\right]=1.$$

Randomness: *For all $\mathsf{m}\in\mathcal{M}$, let $\mathsf{ss}\xleftarrow{\$}\mathsf{SS}_{k-1}.\mathsf{Share}(\mathsf{m})$ the following distributions are identical:*

$$\begin{aligned}&\left\{\mathsf{ss}\middle|\,\mathsf{ss}:=\mathsf{SS}_k.\mathsf{Share}(\mathsf{m})\,\right\}\\ \equiv&\left\{\mathsf{ss}\middle|\begin{array}{l}\mathsf{ss}':=\mathsf{SS}_k.\mathsf{Share}(\mathsf{m})\\ \textbf{for } i\in[\mathsf{n}]:\\ \quad r'_i\hookleftarrow\mathsf{U}(\mathcal{M});\quad \mathsf{ss}''_i\xleftarrow{\$}\mathsf{SS}_k.\mathsf{Share}(r'_i)\\ \quad \mathsf{ss}:=\mathsf{Fold}\circ\mathsf{Proj}\circ\mathsf{Rotate}\circ\mathsf{Lift}(\mathsf{ss}',\mathsf{ss}''_1,\ldots,\mathsf{ss}''_{\mathsf{t}})\end{array}\right\}.\end{aligned}$$

In our layered MPC protocol, we employ dimension projection to transmit shares between consecutive layers. While this strategy effectively ensures corruption cohesion, it reduces the dimension of the shares at each layer. Consequently, the sender must start with shares of a sufficiently high dimension to ensure they remain valid (i.e., have a dimension of at least one) upon reaching the final layer. This naïve approach, however, leads to a communication complexity that grows

exponentially with the number of layers. To overcome this, we introduce the dimension lifting operation. This operation allows parties to locally increase the dimension of their shares, enabling the maintenance of a constant share dimension across layers without incurring additional communication costs.

5.3 Instantiation with Classic Robust Secret Sharing Schemes

Definition 6 (k-dimensional Shamir's RSS). *A k-dimensional (n,t)-Shamir secret sharing scheme is defined recursively as in Definition 3, where the 1-dimensional (n,t)-Shamir secret sharing scheme is used as the base case. Equivalently, we define the k-dimensional Shamir's RSS as follows:*

$\mathsf{Share}(m)$: *Given a message $\mathsf{m} \in \mathcal{M}$:*

1. *Samples a polynomial $\mathsf{F} \in \mathcal{M}[X_k, \ldots, X_1]$ of degree t in each variable uniformly at random, such that $\mathsf{F}(0, \ldots, 0) = \mathsf{m}$.*
2. *For each $i \in [\mathsf{n}]$, sets the i-th share as $[\![\mathsf{m}]\!][i] = \mathsf{F}(i, X_{k-1}, \ldots, X_1)$.*
3. *Outputs the shares $([\![\mathsf{m}]\!][1], \ldots, [\![\mathsf{m}]\!][\mathsf{n}])$.*

$\mathsf{Reconst}([\![\mathsf{m}]\!][1], \ldots, [\![\mathsf{m}]\!][\mathsf{n}])$:

1. *Using the shares, runs the Berlekamp-Welch algorithm [BW86] to recover a polynomial $\mathsf{F} \in \mathcal{M}[X_k, \ldots, X_1]$ of degree t in each variable that agrees with at least $\mathsf{n} - \mathsf{t} \geq 2\mathsf{t} + 1$ shares.*
2. *If such a polynomial does not exist, outputs $\perp$. Otherwise, outputs $\mathsf{m} := \mathsf{F}(0, \ldots, 0)$.*

The correctness, robustness, secrecy, and additive homomorphic properties of the k-dimensional Shamir's RSS follow directly from the properties of the 1-dimensional Shamir's RSS. The strong robustness property follows from the Berlekamp-Welch algorithm [BW86], which can efficiently identify and correct up to t erroneous shares.

Definition 7 (Projection for $\mathsf{SS}^{\mathsf{Shamir}}$). *Let $\mathsf{ss} \xleftarrow{\$} \mathsf{SS}_k^{\mathsf{Shamir}}.\mathsf{Share}(\mathsf{m})$, we define the projection operation $\mathsf{Proj}(\mathsf{ss})$ as follows:*

$\mathsf{Proj}(\mathsf{ss})$*: Parse ss as $\mathsf{F}(X_k, \ldots, X_1)$ as input, then output*

$$\mathsf{ss}[i] = \mathsf{F}(i, X_{k-1}, \ldots, X_1).$$

$\mathsf{Fold}(\mathsf{ss}[1], \ldots, \mathsf{ss}[\mathsf{n}])$*: Takes as input*

$$\mathsf{ss} = (\mathsf{ss}[1], \ldots, \mathsf{ss}[\mathsf{n}]) = (\mathsf{F}(1, X_{k-1}, \ldots, X_1), \ldots, \mathsf{F}(\mathsf{n}, X_{k-1}, \ldots, X_1)).$$

Compute the corresponding polynomial $\mathsf{F}(X_k, \ldots, X_1)$ using the Berlekamp-Welch algorithm and outputs $\mathsf{ss}' = \mathsf{F}(0, X_{k-1}, \ldots, X_1)$.

The correctness of the projection operation follows directly from the definition. The secrecy-preserving property follows from the fact that each share in the projected dimension is a Shamir secret sharing of the corresponding share in the original dimension. We give a proof of the secrecy-preserving property in Section C.

Definition 8 (Lifting for $\mathsf{SS}^{\mathsf{Shamir}}$). *The lifting operation* Lift *takes a* $(k-1)$*-dimensional share* $[\![\mathsf{m}]\!]$ *of a message* m*, and a vector of* t *many* $(k-1)$*-dimensional shares* $[\![r_1]\!], \ldots, [\![r_\mathsf{t}]\!]$ *of random messages as input, and outputs a* k*-dimensional secret share* ss' *as follows:*

$\mathsf{Lift}([\![\mathsf{m}]\!], [\![r_1]\!], \ldots, [\![r_\mathsf{t}]\!])$: *Parse* $[\![\mathsf{m}]\!]$ *as* $\mathsf{F}_0(i, X_{k-1}, \ldots, X_1)$*, and for each* $j \in [\mathsf{t}]$*, parse* $[\![r_j]\!]$ *as* $\mathsf{F}_j(i, X_{k-1}, \ldots, X_1)$*. Using the Lagrange interpolation to find the unique degree* t *polynomial* $\mathsf{F}(X_k)$ *such that* $\mathsf{F}(0) = \mathsf{F}_0$ *and* $\mathsf{F}(\mathsf{t}+1) = \mathsf{F}_1, \ldots, \mathsf{F}(2\mathsf{t}) = \mathsf{F}_\mathsf{t}$*, output* F *as output.*

More precisely, we define the dimension lifting protocol in Tab. 16.

We give the fully instantiation of the k-dimensional CNF RSS in Section E.

6 Messaging Across Layers - Future Messaging and Resharing

A key functionality in the layered setting is securely sending messages between non-adjacent layers, i.e., from a party in layer $\mathcal{L}_i$ to a party in layer $\mathcal{L}_j$ where $j > i+1$. While standard MPC models provide this capability directly through secure channels, [DDG+23] transforms the computation model to layered circuits to avoid this problem. To bridge this gap, we introduce three distinct cross-layer messaging functionalities. These will serve as fundamental building blocks for our constructions, each tailored for different scenarios.

The ideal functionality for future messaging provided in Section A.

We introduce Honest Future Messaging firstly, a variant of this functionality, which guarantees correct message delivery, but relies on the assumption that both the sender and the receiver are honest.

6.1 Honest future messaging

This is the simplest cross-layer messaging functionality, which allows a party in layer $\mathcal{L}_i$ to send a message to a party in layer $\mathcal{L}_j$ where $j > i+1$, with the guarantee that the message will be delivered correctly as long as both the sender and receiver are honest. To send a message m across d layers, the sender embeds m into a d-dimensional robust secret sharing, then successively folds and projects it layer by layer to the next layer. Privacy holds trivially when both the sender and the receiver are honest. Note that this method is suitable only for small d, since the lifting dimension grows with the transmission distance. We provide the details in Tab. 1 using the multi-dimensional robust secret sharing as described in Definition 2.

Tab. 1: Honest Future Messaging Π_{HFM}

Setup:
Input Party: $\mathsf{S} \in \mathcal{L}_0$. Without loss of generality, we denote the sender's layer as $\mathcal{L}_0$.
Output Parties: $\mathsf{R} \in \mathcal{L}_{d+1}$, for some positive integer $d > 0$.

Protocol parameters
- d: a positive integer, the number of layers between the sender and the receiver.
- SS_d: a d-dimensional (n, t)-robust secret sharing scheme with privacy-preserving projection Proj.

Program execution:

Pseudocode of $\mathbf{\Pi}_{\mathsf{HFM}}$:

Layer $\mathcal{L}_0$: S proceeds as follows
1. **recv** m **from** S **in** $\mathcal{L}_0$
2. $\quad \mathsf{ss}_d \xleftarrow{\$} \mathsf{SS}_d.\mathsf{Share}(\mathsf{m})$
3. $\quad$ **for** $k \in [\mathsf{n}]$: S **send** $\mathsf{ss}_d[k]$ **to** P_k^1 **in** $\mathcal{L}_1$

Layer $\mathcal{L}_1$: P_k^1 proceeds as follows
1. **recv** $\mathsf{ss}_d[k]$ **from** S **in** $\mathcal{L}_0$
2. $\quad (\mathsf{ss}_{d-1}[k,1], \ldots, \mathsf{ss}_{d-1}[k,\mathsf{n}]) := \mathsf{Proj}(\mathsf{ss}_d[k])$
3. $\quad$ **for** $k' \in [\mathsf{n}]$:
4. $\quad\quad$ **send** $\mathsf{ss}_{d-1}[k,k']$ **to** $\mathsf{P}_{k'}^2$ **in** $\mathcal{L}_2$

Layer $\mathcal{L}_\ell$ with $\ell \in [2, d-1]$: P_k^ℓ proceeds as follows
1. **recv** $\mathsf{ss}'_{d+1-\ell}[k'',k]$ **from** $\mathsf{P}_{k''}^{\ell-1}$ **in** $\mathcal{L}_{\ell-1}$
2. $\quad \mathsf{ss}_{d+1-\ell}[k] := \mathsf{Fold}(\mathsf{Rotate}(\mathsf{ss}'_{d+1-\ell}[1,k], \ldots, \mathsf{ss}'_{d+1-\ell}[\mathsf{n},k]))$
3. $\quad (\mathsf{ss}_{d-\ell}[k,1], \ldots, \mathsf{ss}_{d-\ell}[k,\mathsf{n}]) := \mathsf{Proj}(\mathsf{ss}_{d+1-\ell}[k])$
4. $\quad$ **for** $k' \in [\mathsf{n}]$:
5. $\quad\quad$ **send** $\mathsf{ss}_{d-\ell}[k,k']$ **to** $\mathsf{P}_{k'}^{\ell+1}$ **in** $\mathcal{L}_{\ell+1}$

Layer $\mathcal{L}_d$: P_k^d proceeds as follows
1. **recv** $\mathsf{ss}'_1[k'',k]$ **from** $\mathsf{P}_{k''}^{d-1}$ **in** $\mathcal{L}_{d-1}$
2. $\quad \mathsf{ss}_1[k] := \mathsf{Fold}(\mathsf{Rotate}(\mathsf{ss}'_1[1,k], \ldots, \mathsf{ss}'_1[\mathsf{n},k]))$
3. $\quad$ **send** $\mathsf{ss}_1[k]$ **to** R **in** $\mathcal{L}_{d+1}$

Layer $\mathcal{L}_{d+1}$: R proceeds as follows
1. **recv** $\mathsf{ss}_1[1], \ldots, \mathsf{ss}_1[\mathsf{n}]$ **from** $\mathsf{P}_1^d, \ldots, \mathsf{P}_\mathsf{n}^d$ **in** $\mathcal{L}_d$
2. $\quad \mathsf{m} := \mathsf{Fold}(\mathsf{ss}_1[1], \ldots, \mathsf{ss}_1[\mathsf{n}])$

6.2 Honest Future Sharing

A variant of honest future messaging, which we call Honest Future Sharing, is designed to enable a party in an early layer to have a message secret-shared among all parties in a distant future layer. In short, the sender simply lifts the message by one additional dimension compared with Future Messaging, then proceeds exactly as in Future Messaging. Consequently, every party in the receiving layer holds a share of the message. The details of Honest Future Sharing($\mathbf{\Pi}_{\mathsf{HFS}}$) is provided in Section D.

6.3 Future Resharing

A core principle of our construction is to maintain secrets in a shared state across all layers. This necessitates a mechanism to transfer a secret sharing from parties in one layer to those in a subsequent layer. We introduce a primitive for this purpose, which we call Future Resharing. Unlike a single party holding the secret(as in future messaging), the secret is already shared among parties in the source layer. The goal is to generate a new, independent sharing of the same secret among parties in a future destination layer. The formal construction is provided in Tab. 2.

Tab. 2: Future Resharing Π_{FRS}

Setup:
Input Parties: Parties in $\mathcal{L}_0$ hold 1-dimensional shares of m.
Output Parties: Parties in $\mathcal{L}_{d+1}$, for some positive integer $d > 0$.
Protocol parameters
- d: a positive integer, the target layer of future resharing.

Preprocess: For each $i \in [0, d]$, each P_k^i receives the shares of C random values(i.e., $[\![r_{i,1}]\!][k], \ldots, [\![r_{i,\mathsf{C}}]\!][k]$). Specifically, $\mathsf{C} = \mathsf{t}$ for Shamir-based secret sharing and $\mathsf{C} = \mathsf{M}$ for CNF-based secret sharing.

Program execution:
Pseudocode of Π_{FRS}:
Layer $\mathcal{L}_0$: P_k^0 proceeds as follows
1. **recv** $[\![\mathsf{m}]\!][k], [\![r_{0,1}]\!][k], \ldots, [\![r_{0,\mathsf{C}}]\!][k]$ **in** $\mathcal{L}_0$
2. $\mathsf{ss}_2[k] := \mathsf{Lift}([\![\mathsf{m}]\!][k], [\![r_{0,1}]\!][k], \ldots, [\![r_{0,\mathsf{C}}]\!][k])$
3. $(\mathsf{ss}_1[k, 1], \ldots, \mathsf{ss}_1[k, \mathsf{n}]) := \mathsf{Proj}(\mathsf{ss}_2[k])$
4. **for** $k' \in [\mathsf{n}]$:
5. **send** $\mathsf{ss}_1[k, k']$ **to** $\mathsf{P}_{k'}^1$ **in** $\mathcal{L}_1$

Layer $\mathcal{L}_\ell$ with $\ell \in [1, d]$: P_k^ℓ proceeds as follows
1. **recv** $\mathsf{ss}_1[k'', k]$ **from** $\mathsf{P}_{k''}^{\ell-1}$ and $[\![r_{\ell,1}]\!][k], \ldots, [\![r_{\ell,\mathsf{C}}]\!][k]$ **in** $\mathcal{L}_{\ell-1}$
2. $[\![\mathsf{m}]\!][k] := \mathsf{Fold}(\mathsf{Rotate}(\mathsf{ss}_1[1, k], \ldots, \mathsf{ss}_1[\mathsf{n}, k]))$
3. $\mathsf{ss}_2[k] := \mathsf{Lift}([\![\mathsf{m}]\!][k], [\![r_{\ell,1}]\!][k], \ldots, [\![r_{\ell,\mathsf{C}}]\!][k])$
4. $(\mathsf{ss}_1[k, 1], \ldots, \mathsf{ss}_1[k, \mathsf{n}]) := \mathsf{Proj}(\mathsf{ss}_2[k])$
5. **for** $k' \in [\mathsf{n}]$:
6. **send** $\mathsf{ss}_1[k, k']$ **to** $\mathsf{P}_{k'}^{\ell+1}$ **in** $\mathcal{L}_{\ell+1}$

Layer $\mathcal{L}_{d+1}$: P_k^{d+1} proceeds as follows
1. **recv** $\mathsf{ss}_1[k'', k]$ **from** $\mathsf{P}_{k''}^d$ **in** $\mathcal{L}_d$
2. $[\![\mathsf{m}]\!][k] := \mathsf{Fold}(\mathsf{Rotate}(\mathsf{ss}_1[1, k], \ldots, \mathsf{ss}_1[\mathsf{n}, k]))$

7 Generic Layered VSS Protocol

We give our construction of generic layered VSS protocol $\mathbf{\Pi}_{\mathsf{VSS}}$ in Tab. 3. The protocol is parameterized by three k-dimensional (n, t)-robust secret sharing schemes SS_k for $k = 1, 2, 3, 4$. The protocol proceeds in four layers: the dealer in $\mathcal{L}_0$, internal computation layers $\mathcal{L}_1, \ldots, \mathcal{L}_4$, and the shareholders in $\mathcal{L}_5$.

Overall Structure The protocol proceeds through the following phases:

$\mathsf{D} \to \mathcal{L}_1$**:** The dealer shares the secret using SS_4 and sends the resulting shares to $\mathcal{L}_1$.

$\mathcal{L}_1 \to \mathcal{L}_2$**:** Each party in $\mathcal{L}_1$ projects the received shares and forwards them to $\mathcal{L}_2$.

$\mathcal{L}_2 \to \mathcal{L}_3$**:** Each party in $\mathcal{L}_2$ performs error detection on the projections received from $\mathcal{L}_1$. Upon detecting an error, the party broadcasts the index of the erroneous share. Subsequently, each party in $\mathcal{L}_2$ projects the shares and sends them to $\mathcal{L}_3$. When P_j^2 detects that P_i^1 is inconsistent, all parties deduce that at least one of P_i^1, P_j^2, or D is corrupted.

$\mathcal{L}_3 \to \mathcal{L}_4$**:** Leveraging the multi-dimensional structure of the shares, every party in $\mathcal{L}_3$ can locally repair the projections of shares delivered from both the D and $\mathcal{L}_1$ without additional communication, assuming the dealer is honest. Each party in $\mathcal{L}_3$ then broadcasts the repaired shares, enabling all parties

to determine whether $\mathcal{L}_1$ or the D is corrupted or whether a $\mathcal{L}_2$ party is maliciously raising false inconsistencies. Finally, each party in $\mathcal{L}_3$ projects the shares and sends them to $\mathcal{L}_4$.

$\mathcal{L}_4 \to \mathcal{L}_5$**:** Assuming the dealer is honest, every party in $\mathcal{L}_4$ can locally repair the projected shares delivered from the D to $\mathcal{L}_1$ without additional communication. Each party in $\mathcal{L}_4$ broadcasts the repaired shares for parties in $\mathcal{L}_1$ identified as corrupted in the previous phase. Finally, each party in $\mathcal{L}_4$ projects the shares and sends them to $\mathcal{L}_5$. After incorporating the broadcast messages from $\mathcal{L}_4$ regarding $\mathcal{L}_1$, every honest party obtains a consistent view of the shares of $\mathcal{L}_1$.

$\mathcal{L}_5$**:** Since every honest party in $\mathcal{L}_1$ has a now-consistent share and there are at least $\mathsf{n} - \mathsf{t}$ such parties, every party in $\mathcal{L}_5$ can locally decode to recover the secret share.

Complexity Optimization A straightforward implementation of the above strategy requires a 5-dimensional robust secret sharing scheme, resulting in communication complexity of $O(\mathsf{n}^5)$ when instantiated with Shamir secret sharing. This is because each layer contributes a factor of n to the overall complexity by consuming a distinct variable. We optimize this by observing that in $\mathcal{L}_4$, parties in $\mathcal{L}_3$ can directly distribute their shares to all parties in $\mathcal{L}_4$ without projecting them, thereby reducing communication overhead. This optimization trades the structured information of shares in $\mathcal{L}_4$ for improved efficiency. This trade-off is acceptable since $\mathcal{L}_4$ serves only as a relay layer for broadcasting messages and does not require decoding. Consequently, this optimization reduces the protocol's structural communication complexity from n^5 to n^4.

Tab. 3: Generic Verifiable Secret Sharing Protocol Π_{VSS}

Protocol Description:

Participants: Dealer $\mathsf{D} \in \mathcal{L}_0$, shareholders in $\mathcal{L}_5$ and internal computation layers $\mathcal{L}_1, \ldots, \mathcal{L}_4$.
Secret Inputs: D inputs $\mathsf{m} \in \mathcal{M}$.
Sub-protocols: Protocol $\mathsf{SS}_4, \mathsf{SS}_3, \mathsf{SS}_2, \mathsf{SS}_1$. For clarity, we denote by Proj_i, Fold_i the projection and folding operations for the dimension i.

Protocol Execution:

Dealer D:
1. $\mathsf{ss}_4 \xleftarrow{\$} \mathsf{SS}_4.\mathsf{Share}(\mathsf{m})$
2. $(\mathsf{ss}_3[1], \ldots, \mathsf{ss}_3[\mathsf{n}]) := \mathsf{Proj}_i(\mathsf{ss}_4)$
3. **send** $\mathsf{ss}_3[i]$ **to** P_i^1 **in** $\mathcal{L}_1$

Layer 1: For each $i \in [\mathsf{n}]$, P_i^1 proceeds as follows:
1. **recv** $\mathsf{ss}_3[i]$ **from** D **in** $\mathcal{L}_0$
2. **for** $\mathsf{ErrorDetect}(\mathsf{ss}_3[i]) \neq \emptyset$ **then BCast Complaint**
3. $(\mathsf{ss}_2[i,1], \ldots, \mathsf{ss}_2[i,\mathsf{n}]) := \mathsf{Proj}_j(\mathsf{ss}_3[i])$
4. **for** $j \in [\mathsf{n}]$: **send** $\mathsf{ss}_2[i,j]$ **to** P_j^2 **in** $\mathcal{L}_2$

Layer 2: For each $j \in [\mathsf{n}]$, P_j^2 proceeds as follows:
1. **for** $i \in [\mathsf{n}]$: **recv** $\mathsf{ss}_2[i,j]$ **from** P_i^1 **in** $\mathcal{L}_1$
2. $\mathcal{I}_j := \mathsf{ErrorDetect}(\{\mathsf{ss}_2[i,j]\}_{i\in[\mathsf{n}]})$
3. **BCast** $\mathcal{I}_j$
4. **for** $i \in [\mathsf{n}]$: $(\mathsf{ss}_1[i,j,1], \ldots, \mathsf{ss}_1[i,j,\mathsf{n}]) := \mathsf{Proj}_k(\mathsf{ss}_2[i,j])$
5. **for** $k \in [\mathsf{n}]$: **send** $\{\mathsf{ss}_1[i,j,k]\}_{i\in[\mathsf{n}]}$ **to** P_k^3 **in** $\mathcal{L}_3$.

Layer 3: For each $k \in [\mathsf{n}]$, P_k^3 proceeds as follows:
1. **for** $j \in [\mathsf{n}]$: **recv** $(\{\mathsf{ss}_1[i,j,k]\}_{i\in[\mathsf{n}]})$ **from** P_j^2 **in** $\mathcal{L}_2$

2. **for** $i,j \in [\mathsf{n}]$: $\mathsf{ss}_1^1[i,j,k] := \begin{cases} \mathsf{ss}_1[i,j,k] & \text{if } i \notin \mathcal{I}_j \\ \bot & \text{if } i \in \mathcal{I}_j \end{cases}$
3. $\{\mathsf{ss}_1^0[i,j,k]\}_{i,j\in[\mathsf{n}]} := \mathsf{Repair}_i(\mathsf{Repair}_j(\{\mathsf{ss}_1^1[i,j,k]\}_{i,j\in[\mathsf{n}]}))$
4. **if** $\mathsf{ss}_1^0[i,j,k] \neq \mathsf{ss}_1^1[i,j,k]$ **then for** $j \in [\mathsf{n}] : \mathcal{J}_k[i,j] = 1$ **else for** $j \in [\mathsf{n}] : \mathcal{J}_k[i,j] = 0$
5. **BCast** $\mathcal{J}_k$
6. **for** $j \in [\mathsf{n}]$: **if** $i \in \mathcal{I}_j$ **BCast** $\{\mathsf{ss}_1^0[i,j,k], \mathsf{ss}_1^1[i,j,k]\}_{i,j\in[\mathsf{n}]}$
7. $\{\mathsf{ss}_1^0[i,k]\}_{i\in[\mathsf{n}]} := \mathsf{Repair}_j(\{\mathsf{ss}_1^0[i,j,k]\}_{j\in[\mathsf{n}]})$
8. $\{\mathsf{ss}_1^1[i,k]\}_{i\in[\mathsf{n}]} := \mathsf{Repair}_j(\{\mathsf{ss}_1^1[i,j,k]\}_{j\in[\mathsf{n}]})$
9. **for** $i \in [\mathsf{n}]$:
10. $\quad (\sigma_{i,k}^0[1], \ldots, \sigma_{i,k}^0[\mathsf{n}]) \xleftarrow{\$} \mathsf{Proj}(\mathsf{SS}_2.\mathsf{Share}(\mathsf{ss}_1^0[i,k]))$
11. $\quad (\sigma_{i,k}^1[1], \ldots, \sigma_{i,k}^1[\mathsf{n}]) \xleftarrow{\$} \mathsf{Proj}(\mathsf{SS}_2.\mathsf{Share}(\mathsf{ss}_1^1[i,k]))$
12. **for** $\ell \in [\mathsf{n}]$: **send** $\{\sigma_{i,k}^0[\ell], \sigma_{i,k}^1[\ell]\}_{i\in[\mathsf{n}]}$ **to** P_ℓ^4 **in** $\mathcal{L}_4$

Public Computation 1:

1. $\mathbf{M} := 0^{\mathsf{n}\times\mathsf{n}\times\mathsf{n}}$; $\mathbf{R} := 0^{\mathsf{n}}$
2. **for** $j \in [\mathsf{n}]$: **if** $i \in \mathcal{I}_j$ **then for** $k \in [\mathsf{n}]$: $\mathbf{M}[i,j,k] := 1$
3. **for** $j \in [\mathsf{n}] \wedge i \in \mathcal{I}_j$:
4. $\quad \mathsf{ss}_\mathsf{p}^0[i,j] := \mathsf{Fold}_k(\{\mathsf{ss}_1^0[i,j,k]\}_{k\in[\mathsf{n}]})$; $\mathsf{ss}_\mathsf{p}^1[i,j] := \mathsf{Fold}_k(\{\mathsf{ss}_1^1[i,j,k]\}_{k\in[\mathsf{n}]})$
5. **if** $\mathsf{ss}_\mathsf{p}^0[i,j] = \bot$ **then abort**
6. **if** $\mathsf{ss}_\mathsf{p}^0[i,j] \neq \mathsf{ss}_\mathsf{p}^1[i,j]$ **then**
7. $\quad$ **for** $j,k \in [\mathsf{n}] : \mathbf{M}[i,j,k] := 1$
8. **if Complaint** is broadcast from P_i^1 **then**
9. $\quad$ **for** $j,k \in [\mathsf{n}] : \mathbf{M}[i,j,k] := 1$
10. **for** $i,j,k \in [\mathsf{n}] : \mathbf{M}[i,j,k] := 1$ **if** $\mathcal{J}_k[i,j] = 1$
11. **if** there are not $2\mathsf{t}+1 \times 2\mathsf{t}+1 \times 2\mathsf{t}+1$ zero submatrix in $\mathbf{M}$ **then abort**
12. **for** $i \in [\mathsf{n}]$: **if** i contains in a $2\mathsf{t}+1 \times 2\mathsf{t}+1 \times 2\mathsf{t}+1$ zero submatrix
13. **then** $\mathbf{R}[i] := 0$ **else** $\mathbf{R}[i] := 1$

Layer 4: For each $\ell \in [\mathsf{n}]$, P_ℓ^4 proceeds as follows:

1. **for** $k \in [\mathsf{n}]$: **recv** $(\{\sigma_{i,k}^0[\ell], \sigma_{i,k}^1[\ell]\}_{i\in[\mathsf{n}]})$ **from** P_k^3 **in** $\mathcal{L}_3$
2. **for** $\mathbf{R}[i] = 1$: **BCast** $\{\sigma_{i,k}^0[\ell]\}_{k\in[\mathsf{n}]}$
3. **for** $i,k \in [\mathsf{n}]$: $(\sigma_{i,k}^1[\ell,1], \ldots, \sigma_{i,k}^1[\ell,\mathsf{n}]) := \mathsf{Proj}_m(\sigma_{i,k}^1[\ell])$
4. **for** $m \in [\mathsf{n}]$: **send** $\{\sigma_{i,k}^1[\ell,m]\}_{i,k\in[\mathsf{n}]}$ **to** P_m^5 **in** $\mathcal{L}_5$

Public Computation 2:

1. **for** $\mathbf{R}[i] = 1$:
2. $\quad \hat{\sigma}_i^0 := \mathsf{Fold}_k(\mathsf{Fold}_\ell(\{\sigma_{i,k}^0[\ell]\}_{k,\ell\in[\mathsf{n}]}))$
3. $\quad$ **if** $\hat{\sigma}_i^0 = \bot$ **then abort**
4. $(\hat{\sigma}_i^0[1], \ldots, \hat{\sigma}_i^0[\mathsf{n}]) := \mathsf{Proj}_m(\hat{\sigma}_i^0)$

Layer 5: For each $m \in [\mathsf{n}]$, P_m^5 proceeds as follows:

1. **for** $\ell \in [\mathsf{n}]$: **recv** $\{\sigma_{i,k}^1[\ell,m]\}_{i,k\in[\mathsf{n}]}$ **from** P_ℓ^4 **in** $\mathcal{L}_4$
2. **for** $i \in [\mathsf{n}]$: $\mathsf{F}'[i,m] := \begin{cases} \hat{\sigma}_i^0[m] & \text{if } \mathbf{R}[i] = 1 \\ \mathsf{Fold}_k(\mathsf{Fold}_\ell(\{\sigma_{i,k}^1[\ell,m]\}_{k,\ell\in[\mathsf{n}]})) & \text{if } \mathbf{R}[i] = 0 \end{cases}$
3. $\hat{\mathsf{F}}[m] := \mathsf{Fold}_i(\mathsf{F}'[1,m], \ldots, \mathsf{F}'[\mathsf{n},m])$
4. Set the share m_m to $\hat{\mathsf{F}}[m]$.

Theorem 1. *Let* SS_k *for* $k \in \{1,2,3,4\}$ *be a* k*-dimensional* (n,t)*-robust secret sharing scheme that supports secrecy-preserving projection. The protocol* $\mathbf{\Pi}_{\mathsf{VSS}}$ *described in Tab. 3 securely implements the* VSS *functionality against an adaptive adversary corrupting up to* t *parties in each layer.*

The detailed proof of Theorem 1 is provided in Section F.

7.1 Instantiation with Shamir-based robust secret sharing

We instantiate the generic VSS protocol in Tab. 3 using the Shamir-based robust secret sharing scheme from Definition 6. The complete protocol is presented in Tab. 4.

Tab. 4: Shamir-based Verifiable Secret Sharing Protocol $\Pi_{\mathsf{VSS}}^{\mathsf{Shamir}}$

Setup:

Participants: Dealer $\mathsf{D} \in \mathcal{L}_0$, shareholders in $\mathcal{L}_5$ and internal computation layers $\mathcal{L}_1, \ldots, \mathcal{L}_4$.
Secret Inputs: D inputs $\mathsf{m} \in \mathcal{M}$.
Sub-protocols:
1. Shamir-based robust secret sharing schemes $\mathsf{SS}_4^{\mathsf{Shamir}}, \mathsf{SS}_3^{\mathsf{Shamir}}, \mathsf{SS}_2^{\mathsf{Shamir}}, \mathsf{SS}_1^{\mathsf{Shamir}}$.
2. Berlekamp-Welch algorithm BW is used to instantiate Fold, $\mathsf{ErrorDetect}$, and Repair for Shamir shares.

Protocol Execution:

Dealer D:
1. Sample a random 4-variate polynomial $\mathsf{F}(W, X, Y, Z)$ of degree at most t in each variable such that $\mathsf{F}(0, 0, 0, 0) = \mathsf{m}$.
2. **for** $i \in [\mathsf{n}]$: **send** $\mathsf{F}(i, X, Y, Z)$ **to** P_i^1 **in** $\mathcal{L}_1$.

Layer 1: For each $i \in [\mathsf{n}]$, P_i^1 proceeds as follows:
1. **recv** $\mathsf{F}(i, X, Y, Z)$ **from** D **in** $\mathcal{L}_0$.
2. **for** $\mathsf{ErrorDetect}(\mathsf{F}(i, X, Y, Z)) \neq \emptyset$ **then BCast Complaint**.
3. **for** $j \in [\mathsf{n}]$: **send** $\mathsf{F}(i, j, Y, Z)$ **to** P_j^2 **in** $\mathcal{L}_2$.

Layer 2: For each $j \in [\mathsf{n}]$, P_j^2 proceeds as follows:
1. **for** $i \in [\mathsf{n}]$: **recv** $\mathsf{F}(i, j, Y, Z)$ **from** P_i^1 **in** $\mathcal{L}_1$.
2. $\mathsf{F}^2(W, j, Y, Z) := \mathsf{BW}_W(\{\mathsf{F}(i, j, Y, Z)\}_{i \in [\mathsf{n}]})$.
3. $\mathcal{I}_j := \{i \mid \mathsf{F}^2(i, j, Y, Z) \neq \mathsf{F}(i, j, Y, Z) \vee \mathsf{F}^2(i, j, Y, Z) = \perp\}$.
4. **BCast** $\mathcal{I}_j$.
5. **for** $k \in [\mathsf{n}]$: **send** $\{\mathsf{F}(i, j, k, Z)\}_{i \in [\mathsf{n}]}$ **to** P_k^3 **in** $\mathcal{L}_3$.

Layer 3: For each $k \in [\mathsf{n}]$, P_k^3 proceeds as follows:
1. **for** $j \in [\mathsf{n}]$: **recv** $\{\mathsf{F}(i, j, k, Z)\}_{i \in [\mathsf{n}]}$ **from** P_j^2 **in** $\mathcal{L}_2$.
2. **for** $i, j \in [\mathsf{n}]$: $\mathsf{F}^1(i, j, k, Z) := \begin{cases} \mathsf{F}(i, j, k, Z) & \text{if } i \notin \mathcal{I}_j \\ \perp & \text{if } i \in \mathcal{I}_j \end{cases}$.
3. $\mathsf{F}^0(W, X, k, Z) := \mathsf{BW}_W(\mathsf{BW}_X(\{\mathsf{F}^1(i, j, k, Z)\}_{i,j \in [\mathsf{n}]}))$.
4. **for** $i, j \in [\mathsf{n}]$: **if** $\mathsf{F}^0(i, j, k, Z) \neq \mathsf{F}^1(i, j, k, Z)$ **then**
5. **for** $j \in [\mathsf{n}]$: $\mathcal{J}_k[i, j] := 1$ **else** $\mathcal{J}_k[i, j] := 0$.
6. **BCast** $\mathcal{J}_k$.
7. **for** $j \in [\mathsf{n}] \wedge i \in \mathcal{I}_j$: **BCast** $\mathsf{F}^0(i, j, k, Z), \mathsf{F}^1(i, j, k, Z)$.
8. **for** $i \in [\mathsf{n}]$: $\mathsf{F}^0(i, 0, k, Z) := (\mathsf{BW}_X(\{\mathsf{F}^0(i, j, k, Z)\}_{j \in [\mathsf{n}]}))(0)$.
9. **for** $i \in [\mathsf{n}]$: $\mathsf{F}^1(i, 0, k, Z) := (\mathsf{BW}_X(\{\mathsf{F}^1(i, j, k, Z)\}_{j \in [\mathsf{n}]}))(0)$.
10. **for** $i \in [\mathsf{n}]$: sample random bivariate polynomials $\mathsf{G}_{i,k}^0(Z, V), \mathsf{G}_{i,k}^1(Z, V)$ of degree at most t such that
11. $\mathsf{G}_{i,k}^0(Z, 0) = \mathsf{F}^0(i, 0, k, Z)$ and $\mathsf{G}_{i,k}^1(Z, 0) = \mathsf{F}^1(i, 0, k, Z)$.
12. **for** $\ell \in [\mathsf{n}]$: **send** $\{\mathsf{G}_{i,k}^0(Z, \ell), \mathsf{G}_{i,k}^1(Z, \ell)\}_{i \in [\mathsf{n}]}$ **to** P_ℓ^4 **in** $\mathcal{L}_4$.

Public Computation 1:
1. $\mathbf{M} := 0^{\mathsf{n} \times \mathsf{n} \times \mathsf{n}}$; $\mathbf{R} := 0^{\mathsf{n}}$.
2. **for** $j \in [\mathsf{n}] \wedge i \in \mathcal{I}_j$: **for** $k \in [\mathsf{n}]$: $\mathbf{M}[i, j, k] := 1$.
3. **for** $j \in [\mathsf{n}] \wedge i \in \mathcal{I}_j$:
4. $\mathsf{H}^0(i, j, Y, Z) := (\mathsf{BW}_Y(\{\mathsf{F}^0(i, j, k, Z)\}_{k \in [\mathsf{n}]}))$.
5. $\mathsf{H}^1(i, j, Y, Z) := (\mathsf{BW}_Y(\{\mathsf{F}^1(i, j, k, Z)\}_{k \in [\mathsf{n}]}))$.
6. **if** $\mathsf{H}^0(i, j, Y, Z) = \perp$ **then abort** .
7. **if** $\mathsf{H}^0(i, j, Y, Z) \neq \mathsf{H}^1(i, j, Y, Z)$ **then for** $j', k \in [\mathsf{n}]$: $\mathbf{M}[i, j', k] := 1$.
8. **if Complaint** is broadcast from P_i^1 **then for** $j, k \in [\mathsf{n}]$: $\mathbf{M}[i, j, k] := 1$.
9. **for** $i, j, k \in [\mathsf{n}]$: **if** $\mathcal{J}_k[i, j] = 1$ **then** $\mathbf{M}[i, j, k] := 1$.
10. **if** there are not $2\mathsf{t} + 1 \times 2\mathsf{t} + 1 \times 2\mathsf{t} + 1$ zero submatrix in $\mathbf{M}$ **then abort** .
11. **for** $i \in [\mathsf{n}]$: **if** i is in a $2\mathsf{t} + 1 \times 2\mathsf{t} + 1 \times 2\mathsf{t} + 1$ zero submatrix **then** $\mathbf{R}[i] := 0$ **else** $\mathbf{R}[i] := 1$.

Layer 4: For each $\ell \in [\mathsf{n}]$, P_ℓ^4 proceeds as follows:
1. **for** $k \in [\mathsf{n}]$: **recv** $\{\mathsf{G}_{i,k}^0(Z, \ell), \mathsf{G}_{i,k}^1(Z, \ell)\}_{i \in [\mathsf{n}]}$ **from** P_k^3 **in** $\mathcal{L}_3$.
2. **for** $\mathbf{R}[i] = 1$: **BCast** $\{\mathsf{G}_{i,k}^0(Z, \ell)\}_{k \in [\mathsf{n}]}$.
3. **for** $m \in [\mathsf{n}]$: **send** $\{\mathsf{G}_{i,k}^1(m, \ell)\}_{i,k \in [\mathsf{n}]}$ **to** P_m^5 **in** $\mathcal{L}_5$.

Public Computation 2:
1. **for** $\mathbf{R}[i] = 1$:
2. $\hat{\sigma}_i^0(Z) := (\mathsf{BW}_Y(\{(\mathsf{BW}_V(\{\mathsf{G}_{i,k}^0(Z, \ell)\}_{\ell \in [\mathsf{n}]}))(0)\}_{k \in [\mathsf{n}]}))(0)$.

3. **if** $\hat{\sigma}_i^0(Z) = \perp$ **then abort** .
4. $(\hat{\sigma}_i^0[1], \ldots, \hat{\sigma}_i^0[\mathsf{n}]) := (\hat{\sigma}_i^0(1), \ldots, \hat{\sigma}_i^0(\mathsf{n}))$.

Layer 5: For each $m \in [\mathsf{n}]$, P_m^5 proceeds as follows:

1. **for** $\ell \in [\mathsf{n}]$: **recv** $\{\mathsf{G}_{i,k}^1(m,\ell)\}_{i,k\in[\mathsf{n}]}$ **from** P_ℓ^4 **in** $\mathcal{L}_4$.
2. **for** $i \in [\mathsf{n}]$: $\mathsf{F}'[i,m] := \begin{cases} \hat{\sigma}_i^0[m] & \text{if } \mathbf{R}[i] = 1 \\ (\mathsf{BW}_Y(\{(\mathsf{BW}_V(\{\mathsf{G}_{i,k}^1(m,\ell)\}_{\ell\in[\mathsf{n}]}))(0)\}_{k\in[\mathsf{n}]}))(0) & \text{if } \mathbf{R}[i] = 0 \end{cases}$.
3. $\hat{\mathsf{F}}[m] := (\mathsf{BW}_W(\{\mathsf{F}'[i,m]\}_{i\in[\mathsf{n}]}))(0)$.
4. Set the share m_m to $\hat{\mathsf{F}}[m]$.

We establish the following theorem regarding the security and efficiency of the Shamir VSS protocol by applying the generic VSS construction from Tab. 3. Therefore, we omit the proof here. Furthermore, to make the protocol's correctness more intuitive, we provide a toy example for Shamir-based VSS when $\mathsf{t} = 1$ and $\mathsf{n} = 4$ in Section G.

Theorem 2 (Shamir-based VSS). *The protocol in Tab. 4 executed in parallel realizes $\mathcal{F}_{\mathsf{Par\text{-}VSS}}$ with perfect t-security for $\mathsf{t} < \mathsf{n}/3$ by consuming 5 layers, and by communicating $O(\mathsf{n}^4 \log |\mathbb{F}|)$ over the point-to-point channel and $O(\mathsf{n}^4 \log |\mathbb{F}|)$ over the broadcast channel for each secret.*

Complexity In terms of point-to-point communication, the Dealer, Layer 1, and Layer 2 each transmit shares of a 4-variate polynomial to the subsequent layer, incurring a complexity of $O(n^4 \log |\mathbb{F}|)$. Although Layer 3 lifts the polynomial to a 5-variate form, the transmission to the next layer involves evaluating one variable at 0, thus maintaining the cost at $O(n^4 \log |\mathbb{F}|)$. Layer 4 similarly incurs a point-to-point cost of $O(n^4 \log |\mathbb{F}|)$. Additionally, Layer 3 and 4 require a broadcast complexity of $O(n^4 \log |\mathbb{F}|)$.

7.2 Instantiation with CNF-based robust secret sharing

We instantiate the generic VSS protocol in Tab. 3 using the CNF-based robust secret sharing scheme from Definition 9. The complete protocol is presented in Tab. 17 in Section H. A key difference is that the CNF-based VSS protocol presents one fewer round than both the Shamir-based variant in Tab. 4 and the generic VSS protocol in Tab. 3.

Round Optimization The CNF-based VSS protocol achieves improved round efficiency compared to the Shamir-based variant through a crucial optimization. Specifically, parties P_k^3 in $\mathcal{L}_3$ can set $\mathcal{I}_j$ as the union of all $\mathcal{I}_j^{\mathsf{c}_2}$ for $\mathsf{c}_2 \in \mathcal{S}_j$. This allows P_k^3 to broadcast the corrected values $\mathsf{m}_{i,j,k,\ell}$ when complaints are raised by parties in $\mathcal{L}_2$. In the Shamir-based VSS protocol, this optimization is not possible because an adversary could raise a complaint against an honest party in $\mathcal{L}_2$, forcing P_ℓ^4 to broadcast shares from honest $\mathcal{L}_1$ parties. By contrast, in the CNF-based VSS protocol, although $\mathcal{I}_j$ may contain some honest parties in $\mathcal{L}_1$, there always exists at least one honest set $\mathcal{S}_i$ such that $\mathcal{I}_j \cap \mathcal{S}_i = \emptyset$. This property, together with CNF-based secret sharing, guarantees perfect secrecy. Consequently,

the CNF-based VSS protocol eliminates one round of communication compared to the Shamir-based variant, thereby improving efficiency while maintaining security.

8 Layered Multiplication Protocol

In this section, we present our layered multiplication protocols. Our primary focus is on a new Shamir-based multiplication protocol. For completeness, we also adapt the CNF-based protocol from [DDG+23] to our setting by replacing its VSS component. The details of the latter are deferred to Section J.

Our Shamir-based protocol is constructed by applying our VSS protocol from Section 7 and cross-layer messaging technique from Section 6 to the BGW multiplication protocol [BGW88], tailored for the layered setting. The resulting protocol uses 8 layers and achieves $O(\mathsf{n}^6)$ complexity. We begin by recalling the standard BGW multiplication protocol.

- **Step 1 (Masking).** Each party P_i holds shares $[\![a]\!][i]$ and $[\![b]\!][i]$ of secrets a and b. P_i defines $[\![c]\!][i] = [\![a]\!][i] \cdot [\![b]\!][i]$. It then masks these values by polynomials $f_i(x)$ and $g_i(x)$ with random coefficients and constant terms $[\![a]\!][i]$ and $[\![b]\!][i]$ respectively. Let $h_i(x) = f_i(x) \cdot g_i(x)$, which is a degree-$2t$ polynomial.

$$f_i(x) = [\![a]\!][i] + \sum_{l=1}^{\mathsf{t}} \alpha_{i,l} x^l \qquad g_i(x) = [\![b]\!][i] + \sum_{l=1}^{\mathsf{t}} \beta_{i,l} x^l$$

$$h_i(x) = f_i(x) \cdot g_i(x) = [\![c]\!][i] + \sum_{l=1}^{2t} \gamma_{i,l} x^l$$

  P_i then verifiably secret shares all coefficients of $f_i(x)$, $g_i(x)$, and $h_i(x)$ with all other parties.
- **Step 2 (Share Evaluation).** Upon receiving shares of the coefficients from each P_i, every party P_j can locally compute shares of the polynomial evaluations. Specifically, for each $i, k \in [\mathsf{n}]$, P_j computes $[\![f_i(k)]\!][j]$, $[\![g_i(k)]\!][j]$, and $[\![h_i(k)]\!][j]$. For instance:

$$[\![f_i(k)]\!][j] = [\![a]\!][i, j] + \sum_{l=1}^{\mathsf{t}} [\![\alpha_{i,l}]\!][j] k^l$$

  P_j then sends these evaluation shares to the corresponding party P_k.
- **Step 3 (Verification).** Each party P_k reconstructs the values $f_i(k)$, $g_i(k)$, and $h_i(k)$ from the shares received in the previous step. It then verifies for all $i \in [\mathsf{n}]$ that $f_i(k) \cdot g_i(k) = h_i(k)$. If the check fails for some i, P_k broadcasts a complaint $(\textbf{Complaint}, i, k)$.
- **Step 4 (Adjudication).** In response to a complaint $(\textbf{Complaint}, i, k)$, every party P_j broadcasts its shares of the coefficients of $f_i(x)$, $g_i(x)$, and $h_i(x)$. This allows for a public reconstruction and verification of the check from Step 3. If the check still fails, P_i is identified as dishonest.

– **Step 5 (Output).** Each party P_j computes its final share $[\![c]\!][j]$ of the product $c = ab$ by taking the appropriate linear combination of the shares $\{[\![c]\!][i, j]\}$ received from all parties P_i that were not identified as dishonest.

Apart from replacing directly the VSS protocol used in Step 1 with our VSS layered setting in Section 7.1, we identify two main issues when directly adapting the above BGW multiplication protocol to our layered setting to achieve proactive security.

– **Issue 1: Correctness of Shares in Step 2.** In Step 2, each party P_j computes $[\![f_i(k)]\!][j]$, $[\![g_i(k)]\!][j]$, and $[\![h_i(k)]\!][j]$ using the shares it received from each P_i. However, if any P_i is corrupted, it may have shared incorrect values for $[\![a]\!][i]$ or $[\![b]\!][i]$, leading to incorrect computations in Step 2.
– **Issue 2: Cross-Layer Corruption Due to Share Reuse.** The protocol reuses the same shares in multiple steps, specifically between Step 2 and Step 4, and between Step 2 and Step 5. This reuse can lead to vulnerabilities where a corrupted party can exploit the same shares across different layers, potentially compromising the integrity of the protocol.

To address these issues, we employ the honest future resharing technique from Section 6.3 to distribute the shares in Step 2 to new layers. This approach effectively mitigates the risks associated with both identified issues. Specifically:

For Issue 1: The resharing technique allows us to securely reshare the values in Step 2 to new layers, ensuring that even if some parties are corrupted, the integrity of the shares is maintained. By doing so, we can effectively mitigate the risks associated with both identified issues.

For Issue 2: By resharing the values to new layers, we ensure that the shares used in Step 4 and Step 5 are independent of those used in Step 2. This independence prevents a corrupted party from exploiting the same shares across different layers, thereby enhancing the protocol's security.

Putting the above countermeasures together, we obtain our Shamir-based multiplication protocol $\mathbf{\Pi}_{\mathsf{Mul}}^{\mathsf{Shamir}}$, which is detailed in Tab. 5.

Tab. 5: Shamir-based Multiplication $\mathbf{\Pi}_{\mathsf{Mul}}^{\mathsf{Shamir}}$

Setup:

Public Parameters: Input parties $\mathsf{P}_1^0, \ldots, \mathsf{P}_\mathsf{n}^0 \in \mathcal{L}_0$, output parties $\mathsf{P}_1^8, \ldots, \mathsf{P}_\mathsf{n}^8 \in \mathcal{L}_8$.
Secret Input: Each P_i^0 inputs $[\![a]\!][i]$ and $[\![b]\!][i]$.
Sub-protocols:
– Protocol $\mathbf{\Pi}_{\mathsf{RG}}, \mathbf{\Pi}_{\mathsf{FRS}}, \mathbf{\Pi}_{\mathsf{HFM}}, \mathbf{\Pi}_{\mathsf{HFS}}, \mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}$
Preprocess:
– Each P_i^0 in the layer $\mathcal{L}_0$ obtains two sets of random shares $\{[\![r_1]\!][i], \ldots, [\![r_\mathsf{t}]\!][i]\}$ and $\{[\![s_1]\!][i], \ldots, [\![s_\mathsf{t}]\!][i]\}$ via $\mathbf{\Pi}_{\mathsf{RG}}$.

Protocol Execution:

Layer 0: For each $i \in [\mathsf{n}]$, P_i^0 proceeds as follows:
1. Sample $\{\alpha_{i,l}, \beta_{i,l}\}_{l\in[\mathsf{t}]} \xleftarrow{\$} \mathbb{F}$.
2. Define $f_i(x) = [\![a]\!][i] + \sum_{l=1}^{\mathsf{t}} \alpha_{i,l} x^l$, $g_i(x) = [\![b]\!][i] + \sum_{l=1}^{\mathsf{t}} \beta_{i,l} x^l$.
3. Compute $h_i(x) = f_i(x) \cdot g_i(x) = [\![c]\!][i] + \sum_{l=1}^{2t} \gamma_{i,l} x^l$.

4. Share $\{\alpha_{i,l}\}_{l\in[\mathsf{t}]}, \{\beta_{i,l}\}_{l\in[\mathsf{t}]}, [\![c]\!][i], \{\gamma_{i,l}\}_{l\in[2\mathsf{t}]}$ with $\mathcal{L}_3$ via $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}$.
5. $\mathsf{F}_a(i,Y) := \mathsf{Lift}([\![a]\!][i], [\![r_1]\!][i], \ldots, [\![r_\mathsf{t}]\!][i])$
6. $\mathsf{F}_b(i,Y) := \mathsf{Lift}([\![b]\!][i]), [\![s_1]\!][i], \ldots, [\![s_\mathsf{t}]\!][i])$
7. **for** $j \in [\mathsf{n}]$: **send** $\mathsf{F}_a(i,j), \mathsf{F}_b(i,j)$ **to** P_j^5 **in** $\mathcal{L}_5$ via $\mathbf{\Pi}_{\mathsf{HFM}}$

Layer 5: For each $j \in [\mathsf{n}]$, P_j^5 proceeds as follows:

1. **recv** $\{[\![\alpha_{i,l}]\!][j]\}_{l\in[\mathsf{t}]}, \{[\![\beta_{i,l}]\!][j]\}_{l\in[\mathsf{t}]}, [\![c]\!][i,j], \{[\![\gamma_{i,l}]\!][j]\}_{l\in[2\mathsf{t}]}$ **from** P_i^0 via $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}$
2. **recv** $F_a(i,j), F_b(i,j)$ **from** P_i^0 via $\mathbf{\Pi}_{\mathsf{HFM}}$
3. $\hat{\mathsf{F}}_a(X,j) := \mathsf{BW}_X(\{\mathsf{F}_a(i,j)\}_{i\in\mathsf{n}})$
4. $\hat{\mathsf{F}}_b(X,j) := \mathsf{BW}_X(\{\mathsf{F}_b(i,j)\}_{i\in\mathsf{n}})$
5. **for** $i,k \in [\mathsf{n}]$:
6. $\quad [\![f_i(k)]\!][j] := \hat{\mathsf{F}}_a(i,j) + \sum_{l=1}^{\mathsf{t}} [\![\alpha_{i,l}]\!][j] \cdot k^l$
7. $\quad [\![g_i(k)]\!][j] := \hat{\mathsf{F}}_b(i,j) + \sum_{l=1}^{\mathsf{t}} [\![\beta_{i,l}]\!][j] \cdot k^l$
8. $\quad [\![h_i(k)]\!][j] := [\![c]\!][i,j] + \sum_{l=1}^{2\mathsf{t}} [\![\gamma_{i,l}]\!][j] \cdot k^l$
9. $\quad$ **send** $([\![f_i(k)]\!][j], [\![g_i(k)]\!][j], [\![h_i(k)]\!][j])$ **to** P_k^6 **in** $\mathcal{L}_6$
10. **for** $i,k \in [\mathsf{n}]$: Share $[\![f_i(k)]\!][j], [\![g_i(k)]\!][j], [\![h_i(k)]\!][j]$ with $\mathcal{L}_7$ via $\mathbf{\Pi}_{\mathsf{HFS}}$.
11. **for** $i \in [\mathsf{n}]$: Input $[\![c]\!][i,j]$ to $\mathbf{\Pi}_{\mathsf{FRS}}$ to reshare $[\![c]\!][i]$ with $\mathcal{L}_8$.

Layer 6: For each $k \in [\mathsf{n}]$, P_k^6 proceeds as follows:

1. **recv** $\{[\![f_i(k)]\!][j]\}_{i\in[\mathsf{n}]}, \{[\![g_i(k)]\!][j]\}_{i\in[\mathsf{n}]}, \{[\![h_i(k)]\!][j]\}_{i\in[\mathsf{n}]}$ **from** each P_j^5 **in** $\mathcal{L}_5$.
2. **for** $i \in [\mathsf{n}]$:
3. $\quad \hat{f}_i(k) := \mathsf{BW}(\{[\![f_i(k)]\!][j]\}_{j\in[\mathsf{n}]})$
4. $\quad \hat{g}_i(k) := \mathsf{BW}(\{[\![g_i(k)]\!][j]\}_{j\in[\mathsf{n}]})$
5. $\quad \hat{h}_i(k) := \mathsf{BW}(\{[\![h_i(k)]\!][j]\}_{j\in[\mathsf{n}]})$
6. $\quad$ **if** $\hat{f}_i(k) \cdot \hat{g}_i(k) \neq \hat{h}_i(k)$: **BCast** (**Complaint**, i,k)

Layer 7: For each $u \in [\mathsf{n}]$, P_u^7 proceeds as follows:

1. **recv** $\{[\![f_i(k)]\!][j,u]\}_{i,j,k\in[\mathsf{n}]}, \{[\![g_i(k)]\!][j,u]\}_{i,j,k\in[\mathsf{n}]}, \{[\![h_i(k)]\!][j,u]\}_{i,j,k\in[\mathsf{n}]}$ **from** P_j^5 via $\mathbf{\Pi}_{\mathsf{HFS}}$.
2. **for** each (**Complaint**, i,k):
3. $\quad$ **BCast** $\{[\![f_i(k)]\!][j,u]\}_{j\in[\mathsf{n}]}, \{[\![g_i(k)]\!][j,u]\}_{j\in[\mathsf{n}]}, \{[\![h_i(k)]\!][j,u]\}_{j\in[\mathsf{n}]}$

Public computation:

1. **for** each (**Complaint**, i,k):
2. $\quad \hat{f}_i(k) := \mathsf{BW}(\{\mathsf{BW}(\{[\![f_i(k)]\!][j,u]\}_{u\in[\mathsf{n}]})\}_{j\in[\mathsf{n}]})$
3. $\quad \hat{g}_i(k) := \mathsf{BW}(\{\mathsf{BW}(\{[\![g_i(k)]\!][j,u]\}_{u\in[\mathsf{n}]})\}_{j\in[\mathsf{n}]})$
4. $\quad \hat{h}_i(k) := \mathsf{BW}(\{\mathsf{BW}(\{[\![h_i(k)]\!][j,u]\}_{u\in[\mathsf{n}]})\}_{j\in[\mathsf{n}]})$
5. $\quad$ **if** $\hat{f}_i(k) \cdot \hat{g}_i(k) \neq \hat{h}_i(k)$: Mark i discarded.

Layer 8: For each $v \in [\mathsf{n}]$, P_v^8 proceeds as follows:

1. **recv** $\{[\![c]\!][i,v]\}_{i\in[\mathsf{n}]}$ from $\mathcal{L}_5$ via $\mathbf{\Pi}_{\mathsf{FRS}}^{\mathsf{Shamir}}$.
2. Define a set $G \subseteq [\mathsf{n}]$ that for every $i \in G$, i is not discarded.
3. From G, take a subset G' of $2\mathsf{t}+1$ elements.
4. $[\![c]\!][v] = \sum_{i\in G'} \lambda_i [\![c]\!][i,v]$. // λ_i denotes the Lagrange interpolation coefficients.
5. Set $[\![c]\!][v]$ as its share of c.

Theorem 3 (Shamir-based multiplication). *The protocol in Tab. 5 realizes the ideal functionality $\mathcal{F}_{\mathsf{Mult}}$ in the $\mathcal{F}_{\mathsf{VSS}}^{(\mathsf{t},\mathsf{N})}$-hybrid model as defined in Tab. 11 with perfect t-security for $\mathsf{t} < \mathsf{n}/3$ by consuming 6 layers, and by communicating $O(\mathsf{n}^6)$ field elements over the point-to-point channels and $O(\mathsf{n}^6)$ field elements over the broadcast channels.*

Proof. The proof and complexity analysis are provided in Section I. □

9 Layered MPC for Arithmetic Circuit

In this section, we present a layered MPC construction. The construction follows the BGW paradigm and uses Beaver multiplication triples, and the per-gate

layer overhead amortizes to $O(1)$. This paradigm adapts to the layered setting thanks to our future resharing protocol. Our MPC supports both Shamir and CNF secret shares. The protocols of CNF secret shares are provided in Section J.

9.1 Multiplication Gate

To achieve an amortized cost of one layer per multiplication gate, we adopt an offline-online approach based on Beaver triples. In layered setting, pre-generating all triples in a single setup phase is inefficient, as each triple need to be transported to the layer of its corresponding gate, incurring significant overhead.

Instead, we employ a pipelined approach where the generation of a triple is initiated several layers ahead of its consumption. Specifically, to evaluate a multiplication gate at layer $\mathcal{L}_k$, the process of generating the required triple $([\![a]\!], [\![b]\!], [\![ab]\!])$ begins at layer $\mathcal{L}_{k-12}$. The shares of a and b are then forwarded to the necessary layers using our future resharing protocol, $\mathbf{\Pi}_{\mathsf{FRS}}$. This pipelined generation, combined with future resharing, ensures that the triples are available at the correct layer when needed, without introducing cross-layer security vulnerabilities. The full protocol is detailed in Tab. 6.

Complexity During Triple Generation, Step 1 invokes $\mathbf{\Pi}_{\mathsf{RG}}$ twice, incurring a cost of $2 \cdot O(n^5)$. Step 2 invokes $\mathbf{\Pi}_{\mathsf{Mul}}^{\mathsf{Shamir}}$ once, which contributes $O(n^6)$ to both point-to-point and broadcast complexities. Step 3 involves eight invocations of $\mathbf{\Pi}_{\mathsf{FRS}}$. Each $\mathbf{\Pi}_{\mathsf{FRS}}$ calls $\mathbf{\Pi}_{\mathsf{RG}}$ to generate random shares, costing $O(n^5)$, while the execution itself costs only $O(n^2)$. Finally, the execution of the multiplication gate requires a broadcast complexity of only $O(n)$. Consequently, the dominant cost for both point-to-point and broadcast channels is $O(n^6)$, stemming from $\mathbf{\Pi}_{\mathsf{Mul}}^{\mathsf{Shamir}}$.

Tab. 6: Shamir-based Multiplication Gate $\Pi_{\mathsf{Mul\text{-}Gate}}^{\mathsf{Shamir}}$

Setup:

Public Parameters: Input parties $\mathsf{P}_1^k, \ldots, \mathsf{P}_n^k$ in layer $\mathcal{L}_k$ with $k \geq 12$, output parties $\mathsf{P}_1^{k+1}, \ldots, \mathsf{P}_n^{k+1} \in \mathcal{L}_{k+1}$.

Secret Inputs: Each P_i^k gets inputs $[\![x]\!][i]$ and $[\![y]\!][i]$.

Sub-protocols:

- Protocol $\mathbf{\Pi}_{\mathsf{FRS}}, \mathbf{\Pi}_{\mathsf{RG}}, \mathbf{\Pi}_{\mathsf{Mul}}^{\mathsf{Shamir}}$.

Triple Generation:

1. $\mathcal{L}_{k-12}$ invokes $\mathbf{\Pi}_{\mathsf{RG}}$, and each $\mathsf{P}_i^{k-7} \in \mathcal{L}_{k-7}$ get random shares $[\![a]\!][i]$ and $[\![b]\!][i]$.
2. $\mathcal{L}_{k-7}$ invokes $\mathbf{\Pi}_{\mathsf{Mul}}^{\mathsf{Shamir}}$ with input the shares of a and b, and each $\mathsf{P}_i^{k+1} \in \mathcal{L}_{k+1}$ get $[\![ab]\!][i]$.
3. $\mathcal{L}_{k-7}$ invokes $\mathbf{\Pi}_{\mathsf{FRS}}$ to reshare a and b to $\mathcal{L}_{k+1}$, thus $\mathcal{L}_k$ and $\mathcal{L}_{k+1}$ can obtain refresh shares of a and b.

Protocol Execution:

Layer k(Gate Input Layer):

1. Each P_i^k broadcasts $[\![x]\!][i] - [\![a]\!][i]$ and $[\![y]\!][i] - [\![b]\!][i]$.

Layer $k+1$(Gate Output Layer): Each P_j^k proceeds as follows:

1. Obtain $d := \mathsf{BW}(\{[\![x]\!][i] - [\![a]\!][i]\}_{i \in [n]})$ and $d := \mathsf{BW}(\{[\![y]\!][i] - [\![b]\!][i]\}_{i \in [n]})$.
2. Set $[\![xy]\!][j] = d \cdot [\![b]\!][j] + e \cdot [\![a]\!][j] + [\![ab]\!][j] + de$.

9.2 Layered secure MPC Protocol

We now present our layered MPC protocol with perfectly secure and guaranteed output delivery. We work with general arithmetic circuit rather than layered arithmetic circuit.

Suppose each input client P_i^0, with $i \in [\mathsf{n}]$ in the input layer has $x_i \in \mathbb{F}$ as input, and each party in the output layer wants to compute $f(x_1, \ldots, x_\mathsf{n})$. The secure computation of f proceeds in three phases:

Input Sharing Phase Each client verifiably shares their input using the underlying secret sharing scheme. To facilitate the initial transfer of these shares to the first gate's input layer, clients also generate randomness for the future resharing protocol by sampling and verifiably sharing $\mathsf{n} \cdot \mathsf{t}$ random values.

Circuit Evaluation Phase The protocol evaluates the circuit gate by gate. In contrast to standard MPC where static parties hold shares, our layered model uses future resharing to pass shares between layers. Specifically, the layer holding a gate's output shares reshares the secret to the layer responsible for the successor gate's input. This approach allows for the direct evaluation of general arithmetic circuits, bypassing the gate alignment issues present in prior work [DDG+23]. The evaluation of the first gate commences only after its inputs have been delivered to the appropriate layer via future resharing.

Output Reconstruction Phase Finally, the parties in the layer holding the output wire of the final gate reveal their shares to the designated output clients, who then reconstruct the final result.

Theorem 4 (Shamir-based MPC). *Let f be an n-party functionality computed by a general arithmetic circuit C over a finite field $\mathbb{F}$, with D levels and M gates. Then, for any $\mathsf{t} < \mathsf{n}/3$, there is an $(\mathsf{n}, \mathsf{t}, D + 13)$-layered MPC protocol for f in which the communication is $M \cdot O(\mathsf{n}^6)$ field elements over the point-to-point channels and $M \cdot O(\mathsf{n}^6)$ field elements over the broadcast channels.*

Tab. 7: Shamir-based Layered MPC Protocol

Setup:

Public Parameters: A arithmetic circuit C over $\mathbb{F}$ with D levels that computes f. Output layer is $d = D + 13$.
Secret Input: Each input client $\mathsf{P}_i^0, i \in [\mathsf{n}]$ inputs $[\![x]\!]_i$.
Outputs: Each output client receives $f(x_1, \ldots, x_\mathsf{n})$.
Sub-protocols:
- Protocol $\mathbf{\Pi}_{\mathsf{RG}}, \mathbf{\Pi}_{\mathsf{FRS}}, \mathbf{\Pi}^{\mathsf{Shamir}}_{\mathsf{Mul\text{-}Gate}}, \mathbf{\Pi}_{\mathsf{VSS}}$.

Protocol Execution:

Input Sharing Phase:
- $\mathcal{L}_0$ invokes nt $\mathbf{\Pi}_{\mathsf{RG}}{}^1$ in parallel to generate the random numbers required for $\mathbf{\Pi}_{\mathsf{FRS}}$.
- Each P_i^0 invokes $\mathbf{\Pi}_{\mathsf{VSS}}$ to share its inputs x_i to $\mathcal{L}_5$. Then, $\mathcal{L}_5$ invokes $\mathbf{\Pi}_{\mathsf{FRS}}$ to deliver the share to the **Gate Input Layer** of its input wire.

Circuit Evaluation Phase (Starting at $\mathcal{L}_{12}$):
- **Mutiplication Gate.** Each P_i^k inputs $[\![x]\!][i]$ and $[\![y]\!][i]$ at **Gate Input Layer**$(\mathcal{L}_k)$ and invokes $\mathbf{\Pi}^{\mathsf{Shamir}}_{\mathsf{Mul\text{-}Gate}}$, each P_i^{k+1} outputs $[\![xy]\!][i]$ at **Gate Output Layer**$(\mathcal{L}_{k+1})$.
- **Addition Gate.** Each P_i^k inputs $[\![x]\!][i]$ and $[\![y]\!][i]$ at **Gate Input Layer**$(\mathcal{L}_k)$ and each P_i^k outputs $[\![x+y]\!][i] = [\![x]\!][i] + [\![y]\!][i]$ at **Gate Output Layer**$(\mathcal{L}_k)$.
- If the current gate's **Gate Output Layer** does not match its successor's **Gate Input Layer**, invoke $\mathbf{\Pi}_{\mathsf{FRS}}$ to reshare input share to its successor's **Gate Input Layer**.

Output Reconstruction Phase:
- Each P_i^{D+12} in $\mathcal{L}_{D+12}$ reveals its share $[\![y]\!][i]$ on the output wire of the output gate to each output clients in $\mathcal{L}_{D+13}$.
- Each output client in $\mathcal{L}_{D+13}$ recovers $y := \mathsf{BW}(\{[\![y]\!][i]\}_{i\in[n]})$ as the output of the protocol.

Acknowledgments.

This work is supported by the Key R&D Program of Shandong Province, China (Grant 2024ZLGX05), the National Natural Science Foundation of China (Grant 62302272), the Shandong Provincial Outstanding Young Scientists Fund (Overseas) (Grant 2024HWYQ-013) the National Key R&D Program of China (Grant 2022YFB2702800, Grant 2023YFA1011200)

References

AKP20. Benny Applebaum, Eliran Kachlon, and Arpita Patra. The round complexity of perfect mpc with active security and optimal resiliency. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1277–1284. IEEE, 2020.

AL17. Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, 2017.

BEP23. Alexander Bienstock, Daniel Escudero, and Antigoni Polychroniadou. On linear communication complexity for (maximally) fluid mpc. In *Annual International Cryptology Conference*, pages 263–294. Springer, 2023.

BEP25. Alexander Bienstock, Daniel Escudero, and Antigoni Polychroniadou. Perfectly secure fluid mpc with abort and linear communication complexity. *IACR Communications in Cryptology*, 1(4), 2025.

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

BW86. Elwyn R. Berlekamp and Michael E. Welzl. Error correction of algebraic block codes. *US Patent*, (4,633,470), 1986.

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

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

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

[1] Since the adversary may corrupt an arbitrary number of clients, the generation of random shares cannot exploit the Vandermonde matrix trick to reduce communication. We therefore use the same method as in CNF-based secret sharing to generate random shares.

DDG+23. Bernardo David, Giovanni Deligios, Aarushi Goel, Yuval Ishai, Anders Konring, Eyal Kushilevitz, Chen-Da Liu-Zhang, and Varun Narayanan. Perfect MPC over layered graphs. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 360–392. Springer, Cham, August 2023.

DDWY93. Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *J. ACM*, 40(1):17–47, January 1993.

DKLN24. Giovanni Deligios, Anders Konring, Chen-Da Liu-Zhang, and Varun Narayanan. Statistical layered MPC. In *TCC (4)*, volume 15367 of *Lecture Notes in Computer Science*, pages 362–394. Springer, 2024.

DM00. Yevgeniy Dodis and Silvio Micali. Parallel reducibility for information-theoretically secure computation. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 74–92. Springer, Berlin, Heidelberg, August 2000.

GHK+21. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Cham.

GIKR01. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *33rd ACM STOC*, pages 580–589. ACM Press, July 2001.

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

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

KKK08. Jonathan Katz, Chiu-Yuen Koo, and Ranjit Kumaresan. Improving the round complexity of vss in point-to-point networks. In *International Colloquium on Automata, Languages, and Programming*, pages 499–510. Springer, 2008.

KLR06. Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In *Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing*, pages 109–118, 2006.

MR92. Silvio Micali and Phillip Rogaway. Secure computation (abstract). In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 392–404. Springer, Berlin, Heidelberg, August 1992.

OY91. Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks. In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 51–59, 1991.

RS22. Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 719–749. Springer, Cham, August 2022.

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

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

A Ideal Functionalities

We present the three ideal functionalities referenced in the main text as follows:

Tab. 8: Ideal functionality $\mathcal{F}_{\mathsf{Par\text{-}VSS}}$

Setup :
Input parties: $\{\mathsf{S}_1, \ldots, \mathsf{S}_\mathsf{l}\} \in \mathcal{L}_0$
Output parties: $\{\mathsf{R}_1, \ldots, \mathsf{R}_\mathsf{n}\} \in \mathcal{L}_d$
Protocol parameters:
- l, d, k: positive integers, where l is the number of senders, d is the number of layers between the sender and the receiver, and k is the dimension of the robust secret sharing.
- SS_k: a k-dimensional (n, t)-robust secret sharing scheme.

Program execution:
Pseudocode of $\mathsf{M}_{\mathsf{Par\text{-}VSS}}$**:**
1. **recv** $(\mathsf{m}_1, \ldots, \mathsf{m}_\mathsf{l})$ **from** $(\mathsf{S}_1, \ldots, \mathsf{S}_\mathsf{l})$ **in** $\mathcal{L}_0$
2. **for** $i \in [\mathsf{l}]$:
3. **if** $\mathsf{m} = \perp$ **then** $[\![\mathsf{m}]\!] := (\perp, \ldots, \perp)$ **else** $[\![\mathsf{m}]\!] \xleftarrow{\$} \mathsf{SS}_k.\mathsf{Share}(\mathsf{m})$
4. **for** $j \in [\mathsf{n}]$: **send** $[\![\mathsf{m}]\!][j]$ **to** R_j **in** $\mathcal{L}_d$

Tab. 9: Ideal functionality $\mathcal{F}_{\mathsf{RandGen}}$

Setup :
Output parties: $\{\mathsf{R}_1, \ldots, \mathsf{R}_\mathsf{n}\} \in \mathcal{L}_i$
Protocol parameters:
- SS: a (n, t)-robust secret sharing scheme.

Program execution:
Pseudocode of $\mathsf{M}_{\mathsf{RandGen}}$**:**
1. $\mathsf{m} \hookleftarrow \mathsf{U}(\mathcal{M})$; $[\![\mathsf{m}]\!] \xleftarrow{\$} \mathsf{SS}.\mathsf{Share}(\mathsf{m})$
2. **for** $j \in [\mathsf{n}]$: **send** $[\![\mathsf{m}]\!][j]$ **to** R_j **in** $\mathcal{L}_i$

Tab. 10: Ideal functionality $\mathcal{F}_{\mathsf{FM}}$

Setup :
Input parties: $\mathsf{S} \in \mathcal{L}_i$
Output parties: $\mathsf{R} \in \mathcal{L}_j$, where $j = i + d$ for some positive integer d.
Protocol parameters:
- d: a positive integer, the number of layers between the sender and the receiver.

Program execution:
Pseudocode of M_{FM}**:**
1. **recv** m **from** S **in** $\mathcal{L}_i$
2. **wait** until round j
3. **send** m **to** R **in** $\mathcal{L}_j$

A.1 Secure Multiplication

We define the ideal functionality $\mathcal{F}_{\mathsf{Mult}}$ for secure multiplication. This functionality takes as input two sets of secret shares, ss_1 and ss_2, provided by the input layer $\mathcal{L}_0$. It outputs a new set of secret shares, ss_3, such that $\mathsf{Reconst}(\mathsf{ss}_3) = \mathsf{Reconst}(\mathsf{ss}_1) \cdot \mathsf{Reconst}(\mathsf{ss}_2)$. The ideal functionality $\mathcal{F}_{\mathsf{Mult}}$, described in Tab. 11,

operates within the $\mathcal{F}_{\mathsf{VSS}}^{(\mathsf{t},\mathsf{N})}$-hybrid model and maintains internal states $\mathsf{InShare}_1$, $\mathsf{InShare}_2$, and $\mathsf{OutShare}$ to store the input and output secret shares.

Tab. 11: Ideal Functionality $\mathcal{F}_{\mathsf{Mult}}$

Setup:
Parameters: A secret sharing scheme $\mathsf{SS} = (\mathsf{Share}, \mathsf{Reconst})$.
Input phase: Upon receiving the message $(\mathsf{Input}, i, 1, [\![\mathsf{m}_1]\!][i])$ (or $(\mathsf{Input}, i, 2, [\![\mathsf{m}_2]\!][i])$), set $\mathsf{InShare}_1[i] = [\![\mathsf{m}_1]\!][i]$ (or $\mathsf{InShare}_2[i] = [\![\mathsf{m}_2]\!][i]$).

Protocol Execution:
Computation:
1. Compute $\mathsf{m}_1 = \mathsf{SS.Reconst}(\mathsf{InShare}_1[1], \ldots, \mathsf{InShare}_1[\mathsf{n}])$.
2. Compute $\mathsf{m}_2 = \mathsf{SS.Reconst}(\mathsf{InShare}_2[1], \ldots, \mathsf{InShare}_2[\mathsf{n}])$.
3. Compute $\mathsf{m}_3 = \mathsf{m}_1 \cdot \mathsf{m}_2$, and $[\![\mathsf{m}_3]\!] \xleftarrow{\$} \mathsf{SS.Share}(\mathsf{m}_3)$.
4. Sets $\mathsf{OutShare}[i] = [\![\mathsf{m}_3]\!][i]$.
5. Sends $(\mathsf{Ready}, i, \mathsf{OutShare}[i])$ to the output party P_i^d.

A.2 Multi-Party Computation

For a function $f(x_1, \ldots, x_\mathsf{l}) \mapsto (y_1, \ldots, y_\mathsf{m})$, let input layer $\mathcal{L}_0 = (\mathsf{P}_1^0, \ldots, \mathsf{P}_\mathsf{l}^0)$ and output layer $\mathcal{L}_d = (\mathsf{P}_1^d, \ldots, \mathsf{P}_\mathsf{m}^d)$, we define the ideal functionality $\mathcal{F}_{\mathsf{MPC}}^f$ as in Tab. 12. The multi-party computation consists of input stage and computation stage.

Tab. 12: Ideal Functionality $\mathcal{F}_{\mathsf{MPC}}^f$ for MPC

Input stage: Upon receiving $(\mathsf{Input}, \mathsf{P}_\ell^0 \in \mathcal{L}_0, \mathsf{m} \in \mathcal{M})$ from the environment $\mathcal{E}$,
1. Set $\mathsf{Input}[\ell] = \mathsf{m}$.
2. Send Spoke to the input party P_ℓ^0.

Computation Stage:
1. Compute the value of $(y_1, \ldots, y_\mathsf{m}) := f(\mathsf{Input}[1], \ldots, \mathsf{Input}[\mathsf{l}])$.
2. For $m \in \mathcal{L}_d$, set $\mathsf{Output}[m] = y_m$.
3. Upon receiving (Out, ℓ), send $(\mathsf{Computed}, \ell)$ to the output party P_ℓ^d.

B Randomness Generation

Details of Shamir-based randomness generation are given in Tab. 13, Since it invokes VSS n times, the total communication is $\mathsf{n} \cdot \mathsf{Comm}_{\mathsf{VSS}}$.

Tab. 13: Randomness Generation Π_{RG} (Shamir)

Setup:
Input Parties: $\mathsf{P}_1^0, \ldots, \mathsf{P}_\mathsf{n}^0 \in \mathcal{L}_0$.
Output Parties: $\mathsf{P}_1^5, \ldots, \mathsf{P}_\mathsf{n}^5 \in \mathcal{L}_5$.
Public Parameters: A public Vandermonde matrix $M = Van^{\mathsf{n}*(\mathsf{n}-\mathsf{t})}(\beta_1, \cdots, \beta_n)$, where $\beta_1, \cdots, \beta_n$ are mutually-different elements of $\mathcal{M}$.
Sub-protocol: $\Pi_{\mathsf{VSS}}^{\mathsf{Shamir}}$.

Protocol Execution:
Layer 0: For each $i \in [\mathsf{n}]$, P_i^0 proceeds as follows
1. Samples a uniform random value $s_i \xleftarrow{\$} \mathbb{F}$.

2. Invokes a $\mathbf{\Pi}_{\mathsf{VSS}}$ to verifiably share s_i with $\mathcal{L}_3$.

Layer 5: For each $j \in [\mathsf{n}]$, P_j^5 proceeds as follows

1. **recv** $[\![s_i]\!][j]$ **from** P_i^0 via $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}$.
2. Compute $[[\![r_1]\!][j], \ldots, [\![r_{\mathsf{n}-\mathsf{t}}]\!][j]]^{\mathrm{T}} = M^{\mathrm{T}}[[\![s_1]\!][j], \ldots, [\![s_{\mathsf{n}}]\!][j]]^{\mathrm{T}}$.
3. Set $[\![r_1]\!][j], \ldots, [\![r_{\mathsf{n}-\mathsf{t}}]\!][j]$ as the share of $\mathsf{n}-\mathsf{t}$ random numbers $r_1, \cdots, r_{\mathsf{n}-\mathsf{t}}$.

Details of CNF-based randomness generation are given in Tab. 14, Since it invokes VSS n times, the total communication is $\mathsf{n} \cdot \mathsf{Comm}_{\mathsf{VSS}}$.

Tab. 14: Randomness Generation Π_{RG} (CNF)

Setup:

Input Parties: $\mathsf{P}_1^0, \ldots, \mathsf{P}_{\mathsf{n}}^0 \in \mathcal{L}_0$.
Output Parties: $\mathsf{P}_1^5, \ldots, \mathsf{P}_{\mathsf{n}}^5 \in \mathcal{L}_5$.
Sub-protocol: $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{CNF}}$.

Protocol Execution:

Layer 0: For each $i \in [\mathsf{n}]$, P_i^0 proceeds as follows

1. Samples a uniform random value $s_i \xleftarrow{\$} \mathbb{G}$.
2. Invokes a $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{CNF}}$ to verifiably share s_i with $\mathcal{L}_3$.

Layer 5: For each $j \in [\mathsf{n}]$, P_j^5 proceeds as follows

1. **recv** $[\![s_i]\!][j]$ **from** P_i^0 via $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{CNF}}$.
2. Compute $[\![r]\!][j] = \sum_{i\in[\mathsf{n}]} [\![s_i]\!][j]$.
3. Set $[\![r]\!][j]$ as the share of the random number $\sum_{i\in[\mathsf{n}]} s_i$.

C The Secrecy-Preserving Property of Projection

C.1 Proof on Shamir's projection

For all $\mathsf{m}, \mathsf{m}' \in \mathcal{M}$, for all $S_k, \ldots, S_1 \subseteq [\mathsf{n}]$ such that $|S_i| < \mathsf{t}$ for all $i \in [k]$, we need to show that the following distributions are identical:

$$\left\{ \{\mathsf{ss}_k[i]\}_{i\in S_k}, \ldots, \{\mathsf{ss}_1[i]\}_{i\in S_1} \;\middle|\; \begin{array}{l} \mathsf{ss}_k \xleftarrow{\$} \mathsf{SS}_k.\mathsf{Share}(\mathsf{m}) \\ \textbf{for } j \in \{k, \ldots, 2\}: \\ \quad \mathsf{ss}_{j-1} := \mathsf{Fold}(\mathsf{Proj}(\mathsf{ss}_j)) \end{array} \right\}$$
$$\equiv \left\{ \{\mathsf{ss}'_k[i]\}_{i\in S_k}, \ldots, \{\mathsf{ss}'_1[i]\}_{i\in S_1} \;\middle|\; \begin{array}{l} \mathsf{ss}'_k \xleftarrow{\$} \mathsf{SS}_k.\mathsf{Share}(\mathsf{m}') \\ \textbf{for } j \in \{k, \ldots, 2\}: \\ \quad \mathsf{ss}'_{j-1} := \mathsf{Fold}(\mathsf{Proj}(\mathsf{ss}'_j)) \end{array} \right\}.$$

We prove this by induction on k.

Case $k = 1$**:** The base case $k = 1$ holds directly from the secrecy property of Shamir's secret sharing.

Case $k - 1 \implies k$**:** Now suppose the statement holds for $k-1$, we prove it for k. Let $[\![\mathsf{m}]\!] = ([\![\mathsf{m}]\!][1], \ldots, [\![\mathsf{m}]\!][\mathsf{n}])$ and $[\![\mathsf{m}']\!] = ([\![\mathsf{m}']\!][1], \ldots, [\![\mathsf{m}']\!][\mathsf{n}])$ be two k-dimensional Shamir's secret sharing of m and m' respectively. For each $i \in [\mathsf{n}]$, let $[\![\mathsf{m}]\!][i] = ([\![\mathsf{m}]\!][i, 1], \ldots, [\![\mathsf{m}]\!][i, \mathsf{n}])$ and $[\![\mathsf{m}']\!][i] = ([\![\mathsf{m}']\!][i, 1], \ldots, [\![\mathsf{m}']\!][i, \mathsf{n}])$ be the $(k-1)$-dimensional shares of $[\![\mathsf{m}]\!][i]$ and $[\![\mathsf{m}']\!][i]$ respectively. Let

$\mathsf{ss} = (\mathsf{ss}[1], \ldots, \mathsf{ss}[\mathsf{n}])$ and $\mathsf{ss}' = (\mathsf{ss}'[1], \ldots, \mathsf{ss}'[\mathsf{n}])$ be the $(k-1)$-dimensional shares obtained by projecting $[\![\mathsf{m}]\!]$ and $[\![\mathsf{m}']\!]$ respectively. For each $j \in [\mathsf{n}]$, let $\mathsf{ss}[j] = (\mathsf{ss}[1, j], \ldots, \mathsf{ss}[\mathsf{n}, j])$ and $\mathsf{ss}'[j] = (\mathsf{ss}'[1, j], \ldots, \mathsf{ss}'[\mathsf{n}, j])$ be the $(k-1)$-dimensional shares of $\mathsf{ss}[j]$ and $\mathsf{ss}'[j]$ respectively. By the definition of projection, we have that for each $j \in [\mathsf{n}]$, $\mathsf{ss}[j]$ and $\mathsf{ss}'[j]$ are 1-dimensional Shamir's secret sharing of $\mathsf{m}_j = \mathsf{SS}_{k-1}.\mathsf{Reconst}([\![\mathsf{m}]\!][j])$ and $\mathsf{m}'_j = \mathsf{SS}_{k-1}.\mathsf{Reconst}([\![\mathsf{m}']\!][j])$ respectively.
Moreover, we have $\mathsf{m} = \mathsf{Reconst}(\{\mathsf{m}_j\}_{j\in[\mathsf{n}]})$ and $\mathsf{m}' = \mathsf{Reconst}(\{\mathsf{m}'_j\}_{j\in[\mathsf{n}]})$. Therefore, by the secrecy property of Shamir's secret sharing, we have that for all $S_k \subseteq [\mathsf{n}]$ such that $|S_k| < \mathsf{t}$, the following distributions are identical:

$$\begin{aligned}&\left\{\{\mathsf{ss}_k[i]\}_{i\in S_k} \middle| \mathsf{ss}_k \xleftarrow{\$} \mathsf{SS}_k.\mathsf{Share}(\mathsf{m})\right\}\\ \equiv &\left\{\{\mathsf{ss}'_k[i]\}_{i\in S_k} \middle| \mathsf{ss}'_k \xleftarrow{\$} \mathsf{SS}_k.\mathsf{Share}(\mathsf{m}')\right\}.\end{aligned}$$

Now, for each $j \in [\mathsf{n}]$, let $\mathsf{ss}_{k-1,j} = (\mathsf{ss}_{k-1,j}[1], \ldots, \mathsf{ss}_{k-1,j}[\mathsf{n}])$ and $\mathsf{ss}'_{k-1,j} = (\mathsf{ss}'_{k-1,j}[1], \ldots, \mathsf{ss}'_{k-1,j}[\mathsf{n}])$ be the $(k-1)$-dimensional shares obtained by folding $\mathsf{ss}[j]$ and $\mathsf{ss}'[j]$ respectively. By the induction hypothesis, we have that for all $S_{k-1} \subseteq [\mathsf{n}]$ such that $|S_{k-1}| < \mathsf{t}$, the following distributions are identical:

$$\begin{aligned}&\left\{\{\mathsf{ss}_{k-1,j}[i]\}_{i\in S_{k-1}} \middle| \begin{array}{l}\mathsf{ss}_{k-1,j} \xleftarrow{\$} \mathsf{SS}_{k-1}.\mathsf{Share}(\mathsf{m}_j)\\ \textbf{for } l \in \{k-1, \ldots, 2\}:\\ \quad \mathsf{ss}_{l-1} := \mathsf{Fold}(\mathsf{Proj}(\mathsf{ss}_l))\end{array}\right\}\\ \equiv &\left\{\{\mathsf{ss}'_{k-1,j}[i]\}_{i\in S_{k-1}} \middle| \begin{array}{l}\mathsf{ss}'_{k-1,j} \xleftarrow{\$} \mathsf{SS}_{k-1}.\mathsf{Share}(\mathsf{m}'_j)\\ \textbf{for } l \in \{k-1, \ldots, 2\}:\\ \quad \mathsf{ss}'_{l-1} := \mathsf{Fold}(\mathsf{Proj}(\mathsf{ss}'_l))\end{array}\right\}.\end{aligned}$$

Combining the two parts, we have that for all $S_k, S_{k-1} \subseteq [\mathsf{n}]$ such that both $|S_k|, |S_{k-1}| < \mathsf{t}$, the following distributions are identical:

$$\begin{aligned}&\left\{\{\mathsf{ss}_k[i]\}_{i\in S_k}, \{\mathsf{ss}_{k-1,j}[i]\}_{i\in S_{k-1}} \middle| \begin{array}{l}\mathsf{ss}_k \xleftarrow{\$} \mathsf{SS}_k.\mathsf{Share}(\mathsf{m})\\ \textbf{for } l \in \{k, \ldots, 2\}:\\ \quad \mathsf{ss}_{l-1} := \mathsf{Fold}(\mathsf{Proj}(\mathsf{ss}_l))\end{array}\right\}\\ \equiv &\left\{\{\mathsf{ss}'_k[i]\}_{i\in S_k}, \{\mathsf{ss}'_{k-1,j}[i]\}_{i\in S_{k-1}} \middle| \begin{array}{l}\mathsf{ss}'_k \xleftarrow{\$} \mathsf{SS}_k.\mathsf{Share}(\mathsf{m}')\\ \textbf{for } lin\{k, \ldots, 2\}:\\ \quad \mathsf{ss}'_{l-1} := \mathsf{Fold}(\mathsf{Proj}(\mathsf{ss}'_l))\end{array}\right\}.\end{aligned}$$

Repeating the above argument for $k-2$ times, we conclude the proof.

C.2 Proof on CNF projection

The secrecy-preserving property of the projection for the CNF scheme follows from its recursive structure and the fundamental properties of the 1-dimensional CNF secret sharing. A k-dimensional CNF share is essentially a k-dimensional matrix of additive secret sharings. At each level of the recursion, a value is split into M additive shares, and each of these shares is then recursively shared at the next dimension.

An adversary corrupting fewer than t parties at any dimension level $j \in [k]$ fails to learn any of the constituent additive shares at that level. This is because, for any set of corrupted parties C with $|C| < \mathsf{t}$, there exists at least one subset $S \in \mathcal{S}$ such that $S \cap C = \emptyset$. The additive sub-secret m_S corresponding to this set S is therefore completely unknown to the adversary. Since the total secret at that level is the sum of all sub-secrets (i.e., $\sum_{S \in \mathcal{S}} \mathsf{m}_S$), and at least one sub-secret remains hidden, the total secret remains hidden. This argument holds recursively through all k dimensions, establishing the secrecy-preserving property.

D Construction of Honest Future Sharing

The protocol of Honest Future Sharing is as follows:

Tab. 15: Honest Future Sharing Π_{HFS}

Setup:

Input Party: $\mathsf{S} \in \mathcal{L}_0$.
Output Parties: Parties in $\mathcal{L}_{d+1}$, for some positive integer $d > 0$.
Protocol parameters
- d: a positive integer, the target layer of honest future sharing.
- SS_d: a d-dimensional (n, t)-robust secret sharing scheme with privacy-preserving projection Proj.

Program execution:

Protocol execution of Π_{HFS}:
Layer $\mathcal{L}_0$: S proceeds as follows
1. **recv** m **from** S **in** $\mathcal{L}_0$
2. $\mathsf{ss}_{d+1} \xleftarrow{\$} \mathsf{SS}_{d+1}.\mathsf{Share}(\mathsf{m})$
3. **for** $k \in [\mathsf{n}]$: S **send** $\mathsf{ss}_{d+1}[k]$ **to** P^1_k **in** $\mathcal{L}_1$

Layer $\mathcal{L}_1$: P^1_k proceeds as follows
1. **recv** $\mathsf{ss}_{d+1}[k]$ **from** S **in** $\mathcal{L}_0$
2. $(\mathsf{ss}_d[k,1], \ldots, \mathsf{ss}_d[k,\mathsf{n}] := \mathsf{Proj}(\mathsf{ss}_{d+1}[k])$
3. **for** $k' \in [\mathsf{n}]$:
4. **send** $\mathsf{ss}_d[k,k']$ **to** $\mathsf{P}^2_{k'}$ **in** $\mathcal{L}_2$

Layer $\mathcal{L}_\ell$ with $\ell \in [2, d-1]$: P^ℓ_k proceeds as follows
1. **recv** $\mathsf{ss}'_{d+2-\ell}[k'',k]$ **from** $\mathsf{P}^{\ell-1}_{k''}$ **in** $\mathcal{L}_{\ell-1}$
2. $\mathsf{ss}_{d+2-\ell}[k] := \mathsf{Fold}(\mathsf{ss}'_{d+2-\ell}[1,k], \ldots, \mathsf{ss}'_{d+2-\ell}[\mathsf{n},k])$
3. $(\mathsf{ss}_{d+1-\ell}[k,1], \ldots, \mathsf{ss}_{d+1-\ell}[k,\mathsf{n}]) := \mathsf{Proj}(\mathsf{ss}_{d+2-\ell}[k])$
4. **for** $k' \in [\mathsf{n}]$:
5. **send** $\mathsf{ss}_{d+1-\ell}[k,k']$ **to** $\mathsf{P}^{\ell+1}_{k'}$ **in** $\mathcal{L}_{\ell+1}$

Layer $\mathcal{L}_d$: P^d_k proceeds as follows
1. **recv** $\mathsf{ss}'_2[k'',k]$ **from** $\mathsf{P}^{d-1}_{k''}$ **in** $\mathcal{L}_{d-1}$
2. $\mathsf{ss}_2[k] := \mathsf{Fold}(\mathsf{ss}'_2[1,k], \ldots, \mathsf{ss}'_2[\mathsf{n},k])$
3. $(\mathsf{ss}_1[k,1], \ldots, \mathsf{ss}_1[k,\mathsf{n}]) := \mathsf{Proj}(\mathsf{ss}_2[k])$
4. **for** $k' \in [\mathsf{n}]$:
5. **send** $\mathsf{ss}_1[k,k']$ **to** $\mathsf{P}^{d+1}_{k'}$ **in** $\mathcal{L}_{d+1}$

Layer $\mathcal{L}_{d+1}$: P^{d+1}_k proceeds as follows
1. **recv** $\mathsf{ss}'_1[k'',k]$ **from** $\mathsf{P}^d_{k''}$ **in** $\mathcal{L}_d$
2. $\mathsf{ss}_1[k] := \mathsf{Fold}(\mathsf{ss}'_1[1,k], \ldots, \mathsf{ss}'_1[\mathsf{n},k])$
3. Set $\mathsf{ss}_1[k]$ as its share of m

E CNF-based multi-dimensional robust secret sharing

Definition 9 (CNF Robust Secret Sharing [ISN89]). *A* (n,t)*-CNF robust secret sharing scheme is a tuple of algorithms* $\mathsf{SS} = (\mathsf{Share}, \mathsf{Reconst})$*, where the message space* $\mathcal{M}$ *is a group and* $\mathsf{n} \geq 3\mathsf{t}+1$*. Let* M *be an enumeration of all* $\binom{\mathsf{n}}{\mathsf{t}}$ *subsets of size* $\mathsf{n}-\mathsf{t}$ *such that* $\mathcal{S} = \{S \subseteq 2^{[\mathsf{n}]} \mid |S| = \mathsf{n}-\mathsf{t}\} = (\mathcal{S}_1, \ldots, \mathcal{S}_{\mathsf{M}})$*. The algorithms are defined as follows:*

$\mathsf{Share}(\mathsf{m})$: *Given a message* $\mathsf{m} \in \mathcal{M}$*:*

1. *Samples* $\{\mathsf{m}_S\}_{S\in\mathcal{S}} \in \mathcal{M}$ *uniformly at random, such that* $\mathsf{m} = \sum_{S\in\mathcal{S}} \mathsf{m}_S$ *(i.e., obtain a k-out-of-k additive secret sharing of* m*).*
2. *For each* $i \in [\mathsf{n}]$*, let* $[\![\mathsf{m}]\!][i] = \{\mathsf{m}_S | S \in \mathcal{S}, i \in S\}$*.*
3. *Outputs the shares* $([\![\mathsf{m}]\!][1], \ldots, [\![\mathsf{m}]\!][\mathsf{n}])$*.*

$\mathsf{Reconst}([\![\mathsf{m}]\!][1], \ldots, [\![\mathsf{m}]\!][\mathsf{n}])$:

1. *For each* $i \in [\mathsf{n}]$ *and* $S \in \mathcal{S}$ *such that* $i \in S$*, denote the additive share* m_S *from* $[\![\mathsf{m}]\!][i]$ *as* m_S^i*. For each* $S \in \mathcal{S}$*, choose the unique value* $\hat{\mathsf{m}}_S$ *such that* $|\{i \in [\mathsf{n}] : \mathsf{m}_S^i = \hat{\mathsf{m}}_S\}| \geq \mathsf{n}-2\mathsf{t}$*.*
2. *Outputs* $\mathsf{m} := \sum_{S\in\mathcal{S}} \hat{\mathsf{m}}_S$*.*

The dimension operation over the CNF robust secret sharing scheme is defined as follows. Noticing that the projection operation over the CNF robust secret sharing scheme is simply the rearrangement of shares, by taking each CNF share and compose them into a vector of shares.

Definition 10 (Projection of $\mathsf{SS}^{\mathsf{CNF}}$). *The projection operation* Proj *for a* k*-dimensional* CNF *share* $[\![\mathsf{m}]\!][i]$ *is defined as follows. Recall that* $[\![\mathsf{m}]\!][i]$ *consists of a set of* $(k-1)$*-dimensional shares* $\{\mathsf{m}_S \mid S \in \mathcal{S}\}$ *with replication. The projection operation simply outputs this set as a vector, indexed by the sets* $S \in \mathcal{S}$ *for which* $i \in S$*.*

We then give the folding and lifting operations over the CNF robust secret sharing scheme as follows.

Definition 11 (Folding of $\mathsf{SS}^{\mathsf{CNF}}$). *The folding operation* $\mathsf{Fold}(\mathsf{ss}[1], \ldots, \mathsf{ss}[\mathsf{M}])$ *takes as input a vector of* k*-dimensional shares and folds them. Moreover, since the* CNF *robust secret share is additively homomorphic, we have*

$\mathsf{Fold}(\mathsf{ss}[1], \ldots, \mathsf{ss}[\mathsf{M}])$: *Computes* $\{m'_S\} = \sum_{i=1}^{\mathsf{M}} \mathsf{ss}[i]$*, and outputs* ss' *such that* $\mathsf{ss}'[i] = \{m'_S \mid i \in S\}$*.*

The composition of Fold, Rotate, and Proj transforms a vector of k-dimensional shares into a vector of $(k-1)$-dimensional shares. For the CNF scheme, these operations align with the structure of the shares. Specifically, let $\mathbf{s}$ be a vector of ℓ k-dimensional CNF shares. Applying Proj to each share in $\mathbf{s}$ yields an $\ell \times \mathsf{M}$ matrix of $(k-1)$-dimensional shares. The Rotate operation transposes this matrix into an $\mathsf{M} \times \ell$ matrix. Finally, applying Fold to the rows of the rotated matrix produces a vector of M $(k-1)$-dimensional shares. We give a proof of the secrecy-preserving property in Section C.

Definition 12 (Lifting of $\mathsf{SS}^{\mathsf{CNF}}$). *We give the lifting protocol* Lift *for the* CNF *robust secret sharing scheme as described in Tab. 16.*

Tab. 16: Dimension Lifting

Setup:
Input Parties: Parties in $\mathcal{L}_0$.
Output Parties: Parties in $\mathcal{L}_0$.
Preprocess: Each P_k^0 receives 1-dimensional $[\![\mathsf{m}]\!][k]$, and the shares of r random values(i.e., $[\![r_1]\!][k], \ldots, [\![r_\mathsf{r}]\!][k]$). Specifically, $\mathsf{r} = \mathsf{t}$ for Shamir-based secret sharing and $\mathsf{r} = \mathsf{M}$ for CNF-based secret sharing.

Program execution:
Pseudocode of Lift**:**
Layer $\mathcal{L}_0$**:** P_k^0 proceeds as follows
- For Shamir-based secret sharing
  1. Using the Lagrange interpolation to find $\mathsf{F}(X)$ such that $\mathsf{F}(0) = [\![\mathsf{m}]\!][k], \mathsf{F}(\mathsf{n}+1) = [\![r_1]\!][k], \ldots, \mathsf{F}(\mathsf{n}+\mathsf{t}) = [\![r_\mathsf{t}]\!][k]$
  2. Set the output 2-dimensional share as $\mathsf{F}(X)$.
- For CNF-based secret sharing
  1. **for** $i \in [\mathsf{M}], k \in \mathcal{S}_i$:
  2. Define $[\![\mathsf{m}_i]\!] := \{\frac{r_{j,i}}{r_{1,i}+\cdots+r_{\mathsf{M},i}} \cdot \mathsf{m}_i\}_{j\in[\mathsf{M}]}$
  3. Set the output 2-dimensional share as $[\![[\![\mathsf{m}]\!][k]]\!] = \{[\![\mathsf{m}_i]\!]\}_{i\in[\mathsf{M}],k\in\mathcal{S}_i}$

F Proof for Generic Layered VSS

We begin the security analysis with a lemma concerning the protocol's complaint mechanism. It establishes that if the dealer is honest, an honest party $\mathsf{P}_i^1 \in \mathcal{L}_1$ is never flagged in the publicly computed vector. Specifically, we have $\mathbf{R}[i] = 0$.

Lemma 1 (Complaint Fairness). *If* D *and* P_i^1 *are honest, then* $\mathbf{R}[i] = 0$ *after the public computation.*

Proof. Let $\widetilde{\mathsf{ss}}_4$ be the 4-dimensional sharing sampled by the honest dealer. For all $k, u, j \in [\mathsf{n}]$, define the canonical values

$$\widetilde{\mathsf{ss}}_3[k] := \mathsf{Proj}(\widetilde{\mathsf{ss}}_4)[k], \qquad \widetilde{\mathsf{ss}}_2[u,k] := \mathsf{Proj}_i(\widetilde{\mathsf{ss}}_3[k])[u],$$
$$\widetilde{\mathsf{ss}}_1[u,j,k] := \mathsf{Proj}_j(\widetilde{\mathsf{ss}}_2[u,k]).$$

Fix any honest P_j^2. For each honest $u \in [\mathsf{n}]$, P_u^1 forwards the correct value $\mathsf{ss}_2[u,j] = \widetilde{\mathsf{ss}}_2[u,j]$. Hence in $\{\mathsf{ss}_2[u,j]\}_{u\in[\mathsf{n}]}$, only corrupted indices u may be faulty, so at most t coordinates are faulty. By strong robustness of $\mathsf{ErrorDetect}$ (Definitions 2 and 4), every honest index is excluded from $\mathcal{I}_j$, in particular $i \notin \mathcal{I}_j$. Therefore $\mathbf{M}[u,j] = 0$ for all honest u, j. Since there are at least $\mathsf{n} - \mathsf{t} \geq 2\mathsf{t}+1$ honest indices, $\mathbf{M}$ contains a $(2\mathsf{t}+1) \times (2\mathsf{t}+1)$ zero submatrix, so the first public computation does not abort.

Now $\mathbf{R}$ is defined in the second public computation. It remains to show: for every j with $i \in \mathcal{I}_j$, we have $\mathsf{ss}_\mathsf{p}^0[i,j] = \mathsf{ss}_\mathsf{p}^1[i,j]$. Fix such j. From the argument above, j must be corrupted.

For each honest $k \in [\mathsf{n}]$, define

$$y_{u,k} := \mathsf{Fold}_{j'}\big(\{\mathsf{ss}_1[u,j',k]\}_{j'\in[\mathsf{n}]}\big) \qquad (u \in [\mathsf{n}]).$$

By protocol syntax, $y_{i,k} = \mathsf{ss}_2^1[i,k]$. For each honest u, only corrupted $\mathcal{L}_2$ indices j' may deviate in the vector $\{\mathsf{ss}_1[u,j',k]\}_{j'}$, so at most t entries are faulty; thus robust folding gives

$$y_{u,k} = \widetilde{\mathsf{ss}}_2[u,k].$$

Therefore

$$\mathsf{ss}_2^1[i,k] = y_{i,k} = \widetilde{\mathsf{ss}}_2[i,k].$$

In Layer 3, the input to Repair_i at honest index k is the i-indexed vector $(y_{1,k},\ldots,y_{\mathsf{n},k})$. This vector differs from the canonical vector $(\widetilde{\mathsf{ss}}_2[1,k],\ldots,\widetilde{\mathsf{ss}}_2[\mathsf{n},k])$ only on corrupted indices u, hence in at most t coordinates. By robust repair (Definition 4),

$$\mathsf{ss}_3^0[k] = \mathsf{Repair}_i(y_{1,k},\ldots,y_{\mathsf{n},k}) = \widetilde{\mathsf{ss}}_3[k].$$

So, for every honest k,

$$\mathsf{ss}_2^0[i,k] = \mathsf{Proj}_i(\mathsf{ss}_3^0[k])[i] = \mathsf{Proj}_i(\widetilde{\mathsf{ss}}_3[k])[i] = \widetilde{\mathsf{ss}}_2[i,k] = \mathsf{ss}_2^1[i,k],$$

and therefore

$$\mathsf{ss}_1^0[i,j,k] = \mathsf{Proj}_j(\mathsf{ss}_2^0[i,k]) = \mathsf{Proj}_j(\mathsf{ss}_2^1[i,k]) = \mathsf{ss}_1^1[i,j,k].$$

Define vectors over $k \in [\mathsf{n}]$:

$$a_k := \mathsf{ss}_1^0[i,j,k], \qquad b_k := \mathsf{ss}_1^1[i,j,k], \qquad c_k := \widetilde{\mathsf{ss}}_1[i,j,k].$$

For every honest k, $a_k = b_k = c_k$. Hence both vectors $(a_k)_{k\in[\mathsf{n}]}$ and $(b_k)_{k\in[\mathsf{n}]}$ differ from $(c_k)_{k\in[\mathsf{n}]}$ in at most t coordinates. Since $(c_k)_{k\in[\mathsf{n}]}$ is valid, robust folding over k returns the same repaired value and is not $\perp$ on both inputs:

$$\mathsf{Fold}_k\big(\{a_k\}_{k\in[\mathsf{n}]}\big) = \mathsf{Fold}_k\big(\{c_k\}_{k\in[\mathsf{n}]}\big) \neq \perp,$$

$$\mathsf{Fold}_k\big(\{b_k\}_{k\in[\mathsf{n}]}\big) = \mathsf{Fold}_k\big(\{c_k\}_{k\in[\mathsf{n}]}\big) \neq \perp .$$

Hence

$$\mathsf{ss}_\mathsf{p}^0[i,j] = \mathsf{ss}_\mathsf{p}^1[i,j].$$

Since the above holds for every checked pair (i,j), the test $\mathsf{ss}_\mathsf{p}^0[i,j] \neq \mathsf{ss}_\mathsf{p}^1[i,j]$ never sets $\mathbf{R}[i]$ to 1. Therefore, $\mathbf{R}[i] = 0$. □

We now present the proof of the main theorem regarding the security of the protocol, followed by lemmas that establish its agreement, strong committing and secrecy properties.

Proof (of Theorem 1). We prove security via the simulation paradigm. We construct a simulator Sim that, for any adversary $\mathcal{A}$, produces a view indistinguishable from a real execution of $\mathbf{\Pi}_{\mathsf{VSS}}$. The simulation strategy depends on whether the dealer is honest or corrupted.

Uncorrupted Dealers P_i^0. We separately discuss the simulator's behavior dealing with the environment and the adversary.

- **Environment Interaction.** The simulator Sim receives the dealer's input m from the ideal functionality $\mathcal{F}_{\mathsf{VSS}}$. It then simulates the honest parties' actions in the protocol, including generating shares, sending messages, and executing the public computation. Since the dealer is honest, all honest parties in $\mathcal{L}_3$ will hold shares of m at the end of the protocol, as guaranteed by the correctness property (Lemma 2). Therefore, Sim instructs $\mathcal{F}_{\mathsf{VSS}}$ to deliver m to all honest parties.
- **Adversary Interaction.** The simulator Sim executes the adversary $\mathcal{A}$, which controls up to t parties in each layer. The simulator uniformly samples a random message $\tilde{\mathsf{m}}_i$, and simulates the honest parties' actions and forwards messages between $\mathcal{A}$ and the honest parties. Since the dealer is honest, the shares held by corrupted parties reveal no information about m, as ensured by the secrecy property (Lemma 10). Thus, the simulated view is indistinguishable from a real execution.

Corrupted Dealer. In this case, the environment does not provide any input to the dealer. The simulator Sim runs the adversary $\mathcal{A}$, which controls the dealer and up to t parties in each layer. The simulator faithfully simulates the honest parties' actions and forwards messages between $\mathcal{A}$ and the honest parties. By the verifiability property (Lemma 3), all honest parties in $\mathcal{L}_3$ will either hold shares of the same message m or the default value 0 at the end of the protocol which is uniquely determined after the public computation by the information held by honest parties. The simulator Sim extracts this message m from the shares held by any honest party in $\mathcal{L}_3$. The simulator then instructs $\mathcal{F}_{\mathsf{VSS}}$ to deliver m to all honest parties.

- If the public computation results in an abort, all honest parties output a default value. Sim instructs the ideal functionality to deliver $\perp$ to all honest parties, which matches the protocol's output.
- If the protocol does not abort, the strong committing property (Lemma 9) guarantees that a unique message m is committed. The simulator can extract this message m from the shares held by any honest party. Sim then provides this m to the ideal functionality. By the verifiability property (Lemma 3), this matches the protocol's outcome, as all honest parties hold shares of the same message m.

In both scenarios, the simulated view is identical to a real-world execution, and the output delivered to the honest parties is consistent with the protocol's outcome. This concludes the proof.

□

Lemma 2 (Correctness). *If the dealer* D *is honest, then all honest parties in* $\mathcal{L}_5$ *hold shares of the dealer's input* $\mathsf{m} \in \mathcal{M}$ *at the end of the protocol.*

Proof. By Lemma 1, any honest party $\mathsf{P}_i^1 \in \mathcal{L}_1$ satisfies $\mathbf{R}[i] = 0$, meaning it is not flagged in the public computation. Since at most t parties in $\mathcal{L}_1$ are corrupted, at least $\mathsf{n} - \mathsf{t}$ honest parties in $\mathcal{L}_1$ remain unflagged. Moreover, there are at least $\mathsf{n} - \mathsf{t}$ honest parties in each dimension of ss_4. When an honest party in $\mathcal{L}_5$ reconstructs by folding the shares received from $\mathcal{L}_4$, at most t shares contain

errors while the remainder are consistent with the honest dealer's sharing. By the error correction property of Fold (Definition 2), the shares computed in $\mathcal{L}_5$ are consistent with the dealer's input m. □

Lemma 3 (Verifiability). *All honest parties in $\mathcal{L}_5$ hold shares of the same message $\mathsf{m} \in \mathcal{M}$ or all $\perp$ at the end of the protocol. Moreover, the output is uniquely determined after the public computation.*

Proof. We distinguish the dealer's behavior.

Honest Dealer. By correctness (Lemma 2), all honest parties in $\mathcal{L}_5$ hold shares of the dealer's input m, so agreement holds.

Corrupted Dealer. Consider the two outcomes of the public computation.

1. **Abort.** If any public check aborts, all honest parties output the default value $\perp$, hence they agree.
2. **No abort.** If the public computation does not abort, then by Lemma 8, all honest parties P^5_m in $\mathcal{L}_5$ output non-$\perp$ values. Moreover, by Lemma 7, for every honest P^5_m, the value $\mathsf{F}'[i,m]$ is consistent with $\mathsf{Proj}_i(\mathsf{Proj}_m(\tau))$ for all honest P^1_i with $\tau \in \mathsf{Supp}(\mathsf{SS}_2.\mathsf{Share})$. Therefore, all honest parties in $\mathcal{L}_5$ hold shares of the same message m determined by these consistent values. Moreover, since the output is determined by the shares held by honest parties in $\mathcal{L}_5$, the output is uniquely determined after the public computation.

This proves both agreement and that the output is uniquely determined once the public computation is fixed. □

Lemma 4 (Honest Row Reconstruction). *If the protocol does not abort, then for every honest P^1_i in $\mathcal{L}_1$ and every honest P^5_m in $\mathcal{L}_5$ such that $\mathbf{R}[i] = 0$, we have*

$$\mathsf{Fold}_k\Big(\mathsf{Fold}_\ell\big(\{\sigma^1_{i,k}[\ell,m]\}_{\ell\in[\mathsf{n}]}\big)\Big) = \mathsf{Proj}_m\Big(\mathsf{Fold}_k\big(\mathsf{Proj}_k(\mathsf{Fold}_j(\mathsf{Proj}_j(\mathsf{ss}_3[i])))\big)\Big)[m].$$

Moreover, we denote by $\mathsf{ss}^H[i,m] = \mathsf{Proj}_m\Big(\mathsf{Fold}_k\big(\mathsf{Proj}_k(\mathsf{Fold}_j(\mathsf{Proj}_j(\mathsf{ss}_3[i])))\big)\Big)[m]$ *the honest reconstruction of the share for P^1_i and P^5_m.*

Proof. Fix honest indices $i, m \in [\mathsf{n}]$ such that $\mathbf{R}[i] = 0$, and assume the protocol does not abort.

Since P^1_i is honest and $\mathbf{R}[i] = 0$, the public rule "if **Complaint** is broadcast from P^1_i, then set $\mathbf{R}[i] := 1$" implies that P^1_i did not broadcast **Complaint**. By Layer 1, an honest P^1_i broadcasts **Complaint** iff $\mathsf{ErrorDetect}(\mathsf{ss}_3[i]) \neq \emptyset$, hence

$$\mathsf{ErrorDetect}(\mathsf{ss}_3[i]) = \emptyset.$$

Define

$$a_{i,j} := \mathsf{Proj}_j(\mathsf{ss}_3[i])[j], \qquad \eta_i := \mathsf{Fold}_j(\{a_{i,j}\}_{j\in[\mathsf{n}]}), \qquad x_{i,k} := \mathsf{Proj}_k(\eta_i)[k].$$

We first determine $\mathsf{ss}_1^1[i,k]$ for an honest P_k^3. For every honest P_j^2, because P_i^1 is honest,

$$\mathsf{ss}_2[i,j] = a_{i,j}, \qquad \mathsf{ss}_1[i,j,k] = \mathsf{Proj}_k(\mathsf{ss}_2[i,j])[k] = \mathsf{Proj}_k(a_{i,j})[k].$$

Since at most t parties in $\mathcal{L}_2$ are corrupted, the vector $\{\mathsf{ss}_1[i,j,k]\}_{j\in[\mathsf{n}]}$ differs from $\{\mathsf{Proj}_k(a_{i,j})[k]\}_{j\in[\mathsf{n}]}$ in at most t positions. Layer 3 sets $\mathsf{ss}_1^1[i,j,k] = \mathsf{ss}_1[i,j,k]$, so robust folding gives

$$\mathsf{ss}_1^1[i,k] = \mathsf{Fold}_j\big(\{\mathsf{ss}_1^1[i,j,k]\}_{j\in[\mathsf{n}]}\big) = \mathsf{Fold}_j\big(\{\mathsf{Proj}_k(a_{i,j})[k]\}_{j\in[\mathsf{n}]}\big).$$

By commutativity of projection and folding (Definition 4),

$$\mathsf{Fold}_j\big(\{\mathsf{Proj}_k(a_{i,j})[k]\}_{j\in[\mathsf{n}]}\big) = \mathsf{Proj}_k\Big(\mathsf{Fold}_j(\{a_{i,j}\}_{j\in[\mathsf{n}]})\Big)[k] = \mathsf{Proj}_k(\eta_i)[k] = x_{i,k}.$$

Hence $\mathsf{ss}_1^1[i,k] = x_{i,k}$.

Now fix this honest k. Layer 3 computes

$$(\sigma_{i,k}^1[1], \ldots, \sigma_{i,k}^1[\mathsf{n}]) := \mathsf{Proj}(\mathsf{SS}_2.\mathsf{Share}(\mathsf{ss}_1^1[i,k])).$$

For each ℓ, define

$$y_{i,k,\ell,m} := \mathsf{Proj}_m(\sigma_{i,k}^1[\ell])[m].$$

Every honest P_ℓ^4 forwards $\sigma_{i,k}^1[\ell,m] = y_{i,k,\ell,m}$. Thus $\{\sigma_{i,k}^1[\ell,m]\}_{\ell\in[\mathsf{n}]}$ differs from $\{y_{i,k,\ell,m}\}_{\ell\in[\mathsf{n}]}$ in at most t positions. By robust folding over ℓ,

$$\begin{aligned}\mathsf{Fold}_\ell\big(\{\sigma_{i,k}^1[\ell,m]\}_{\ell\in[\mathsf{n}]}\big) &= \mathsf{Fold}_\ell\big(\{y_{i,k,\ell,m}\}_{\ell\in[\mathsf{n}]}\big)\\ &= \mathsf{Proj}_m(\mathsf{ss}_1^1[i,k])[m]\\ &= \mathsf{Proj}_m(x_{i,k})[m].\end{aligned}$$

Since $\mathbf{R}[i] = 0$, P_m^5 uses the $\mathbf{R}[i] = 0$ branch and outputs

$$\mathsf{Fold}_k\Big(\mathsf{Fold}_\ell\big(\{\sigma_{i,k}^1[\ell,m]\}_{\ell\in[\mathsf{n}]}\big)\Big).$$

For every honest k, the inner value equals $\mathsf{Proj}_m(x_{i,k})[m]$. Hence only corrupted k can deviate, and robust folding over k yields

$$\mathsf{Fold}_k\Big(\mathsf{Fold}_\ell\big(\{\sigma_{i,k}^1[\ell,m]\}_{\ell\in[\mathsf{n}]}\big)\Big) = \mathsf{Fold}_k\big(\mathsf{Proj}_m(x_{i,k})[m]\big).$$

By commutativity of projection and folding again,

$$\begin{aligned}\mathsf{Fold}_k\big(\mathsf{Proj}_m(x_{i,k})[m]\big) &= \mathsf{Proj}_m\Big(\mathsf{Fold}_k(x_{i,k})\Big)[m]\\ &= \mathsf{Proj}_m\Big(\mathsf{Fold}_k\big(\mathsf{Proj}_k(\mathsf{Fold}_j(\mathsf{Proj}_j(\mathsf{ss}_3[i])))\big)\Big)[m].\end{aligned}$$

This is the claimed equality. □

We define Hon_0 as the set of index i such that P_i^1 is honest and $\mathbf{R}[i] = 0$. and Hon_1 as the set of index i such that P_i^1 is honest and $\mathbf{R}[i] = 1$.

Lemma 5 ($\mathbf{R}[i] = 0$ honest consistency). *If the protocol does not abort, then for every honest P_i^1 in $\mathcal{L}_1$ such that $\mathbf{R}[i] = 0$, there exists $(\mathsf{ss}_i[1], \ldots, \mathsf{ss}_i[\mathsf{n}])$ such that $\mathsf{ErrorDetect}_m(\{\mathsf{ss}_i[m]\}_{m\in[\mathsf{n}]}) = \emptyset$ and for all honest P_m^5 we have $\mathsf{ss}_i[m] = \mathsf{ss}^H[i, m]$.*

Moreover, for every $m \in [\mathsf{n}]$, there exists a valid share $\mathsf{ss}_m[i]$ such that $\mathsf{ErrorDetect}_m(\{\mathsf{ss}_m[i]\}_{i\in[\mathsf{n}]}) = \emptyset$ and for all $i \in \mathsf{Hon}_0$ we have $\mathsf{ss}_i[m] = \mathsf{ss}^H[i, m]$.

Intuitively, Lemma 4 gives us that every P_i^1 with $\mathbf{R}[i] = 0$ has a consistent reconstruction $\mathsf{ss}^H[i, m]$ for every honest P_m^5. The above lemma further states that all these shares $\mathsf{ss}^H[i, m]$ is consistent in dimension i. This follows from the fact that $\mathbf{R}[i] = 0$ implies that there are at least $2\mathsf{t} + 1$ same entries for every $k \in [\mathsf{n}]$ in $\mathbf{M}[i, _, k]$ that are 0. Concretely, it means there are at least $\mathsf{t}+1$ honest parties in $\mathcal{L}_2$ claiming that all their shares received from honest P_i^1 with $\mathbf{R}[i] = 0$ are consistent.

Proof. The first claim follows directly from Lemma 4, as the honest reconstruction $\mathsf{ss}^H[i, m]$ is consistent with the value computed by honest P_m^5 for every honest i with $\mathbf{R}[i] = 0$.

For the second claim, fix any $m \in [\mathsf{n}]$. For every honest $i \in \mathsf{Hon}_0$, by the first claim of this lemma, the value held by honest P_m^5 on row i equals $\mathsf{ss}^H[i, m]$. Moreover, since $\mathbf{R}[i] = 0$, there are at least $2\mathsf{t} + 1$ same entries for every $k \in [\mathsf{n}]$ in $\mathbf{M}[i, _, k]$ that are 0. This means there are at least $\mathsf{t} + 1$ honest parties in $\mathcal{L}_2$ claiming that all their shares received from honest P_i^1 with $\mathbf{R}[i] = 0$ are consistent. Hence, by the error detection property of $\mathsf{ErrorDetect}_m$, there exists a valid share $\mathsf{ss}_m[i]$ such that $\mathsf{ErrorDetect}_m(\{\mathsf{ss}_m[i]\}_{i\in[\mathsf{n}]}) = \emptyset$ and for all $i \in \mathsf{Hon}_0$ we have $\mathsf{ss}_i[m] = \mathsf{ss}^H[i, m]$. □

Lemma 6 ($\mathbf{R}[i] = 1$ consistency). *Assume the protocol does not abort. Then for every $i \in \mathsf{Hon}_1$, i.e. P_i^1 is honest and $\mathbf{R}[i] = 1$, there exists $(\mathsf{ss}_i[1], \ldots, \mathsf{ss}_i[\mathsf{n}])$ such that $\mathsf{ErrorDetect}_m(\{\mathsf{ss}_i[m]\}_{m\in[\mathsf{n}]}) = \emptyset$ and for all honest P_m^5 we have $\mathsf{ss}_i[m] = \mathsf{F}'[i, m]$.*

Moreover, for every honest P_m^5, for all $i \in \mathsf{Hon}_1 \cup \mathsf{Hon}_0$ there exists a valid share $(\mathsf{ss}_m[1], \ldots, \mathsf{ss}_m[\mathsf{n}])$ such that

$$\mathsf{ErrorDetect}_m(\{\mathsf{ss}_m[i]\}_{i\in[\mathsf{n}]}) = \emptyset \quad \textit{and} \quad \mathsf{ss}_m[i] = \mathsf{F}'[i, m] \;\; \forall i \in \mathsf{Hon}_0 \cup \mathsf{Hon}_1.$$

Proof. The first claim follows directly from the protocol syntax, as $\mathsf{F}'[i, m]$ is the value used in the $\mathbf{R}[i] = 1$ branch for honest i and honest m, and the protocol does not abort.

For the second claim, fix an honest P_m^5. For every honest $i \in \mathsf{Hon}_0$, by the first claim of Lemma 5, the value held by honest P_m^5 on row i equals $\mathsf{ss}^H[i, m]$. By the protocol syntax, for the construction of $\mathsf{F}'[i, m]$ for honest $i \in \mathsf{Hon}_1$, the value $\hat{\sigma}_i^0[m]$ is obtained from the public computation and is accepted only in a non-abort execution. Which means the public computed value $\hat{\sigma}_i^0[m]$ is consistent with at least $2\mathsf{t} + 1$ shares provided by P_i^1 such that $\mathbf{R}[i] = 0$ (this is because by the syntax fo the protocol, for all i such that $\mathbf{R}[i] = 1$, their value is set to be $\perp$ in the public computation). Hence, $(\mathsf{ss}_m[1], \ldots, \mathsf{ss}_m[\mathsf{n}])$ is consistent with at least

$\mathsf{t}+1$ honest parties in Hon_0. Moreover Lemma 5, all honest parties in Hon_0 are consistent with the same honest reconstruction $\mathsf{ss}^H[i,m]$. Therefore, by the error detection property of $\mathsf{ErrorDetect}_m$, there exists a valid share $(\mathsf{ss}_m[1],\ldots,\mathsf{ss}_m[\mathsf{n}])$ such that $\mathsf{ErrorDetect}_m(\{\mathsf{ss}_m[i]\}_{i\in[\mathsf{n}]})=\emptyset$ and for all $i\in\mathsf{Hon}_0\cup\mathsf{Hon}_1$ we have $\mathsf{ss}_m[i]=\mathsf{F}'[i,m]$. □

Lemma 7 (Honest Consistency). *If the protocol does not abort, then there exists a valid* 2*-dimensional share* $\tau\in\mathsf{Supp}(\mathsf{SS}_2.\mathsf{Share})$ *such that for every honest* P_i^1 *and honest* P_m^5*, the value* $\mathsf{F}'[i,m]$ *is consistent with* $\mathsf{Proj}_i(\mathsf{Proj}_m(\tau))$*. Formally, for every honest* $i,m\in[\mathsf{n}]$*, we have*

$$\mathsf{F}'[i,m]=\mathsf{Proj}_i(\mathsf{Proj}_m(\tau)).$$

Proof. Assume the protocol does not abort. This lemma is a direct consequence of Lemma 5 and Lemma 6. In particular, by Lemma 5, for every honest P_i^1 such that $\mathbf{R}[i]=0$, there exists a consistent share $\mathsf{ss}^H[i,m]$ for every honest P_m^5. Moreover, by Lemma 6, for every honest P_i^1 such that $\mathbf{R}[i]=1$, there exists a consistent share $\mathsf{F}'[i,m]$ for every honest P_m^5. By the second claim of Lemma 6, for every honest P_m^5, there exists a valid share $(\mathsf{ss}_m[1],\ldots,\mathsf{ss}_m[\mathsf{n}])$ such that $\mathsf{ErrorDetect}_m(\{\mathsf{ss}_m[i]\}_{i\in[\mathsf{n}]})=\emptyset$ and for all $i\in\mathsf{Hon}_0\cup\mathsf{Hon}_1$ we have $\mathsf{ss}_m[i]=\mathsf{F}'[i,m]$. Hence, we can define a valid 2-dimensional share τ such that $\mathsf{Proj}_i(\mathsf{Proj}_m(\tau))=\mathsf{F}'[i,m]$ for every honest $i,m\in[\mathsf{n}]$. □

Lemma 8 (Non-aborting). *If the protocol does not abort, then all honest parties in* $\mathcal{L}_5$ *output a non-*$\perp$ *value at the end of the protocol.*

Proof. By assumption, both public-computation phases pass all abort checks. In particular, every value used in the $\mathbf{R}[i]=1$ branch is well-defined and non-$\perp$.

Let τ be the witness share from Lemma 7, and define

$$\gamma[i,m]:=\mathsf{Proj}_i(\mathsf{Proj}_m(\tau)).$$

By Lemma 7, for each honest P_m^5 and each honest P_i^1:

$$\mathsf{F}'[i,m]=\gamma[i,m].$$

Since at most t indices i are corrupted in $\mathcal{L}_1$, the input vector to

$$\hat{\mathsf{F}}[m]=\mathsf{Fold}_i(\mathsf{F}'[1,m],\ldots,\mathsf{F}'[\mathsf{n},m])$$

differs from the corresponding consistent vector $(\gamma[1,m],\ldots,\gamma[\mathsf{n},m])$ in at most t positions. By robust repair/folding, $\hat{\mathsf{F}}[m]\neq\perp$ for every honest m. Hence every honest party in $\mathcal{L}_5$ outputs a non-$\perp$ value. □

Lemma 9 (Strong Committing). *Even if the dealer* D *is malicious, there exists a unique message* $\mathsf{m}\in\mathcal{M}$ *such that all honest parties in* $\mathcal{L}_5$ *either hold shares of* m *or the default value* $\perp$ *at the end of the protocol.*

Proof. The statement follows directly from Lemma 3. If the protocol aborts, all honest parties in $\mathcal{L}_5$ output $\perp$. Otherwise, all honest parties in $\mathcal{L}_5$ hold shares of one unique message $\mathsf{m} \in \mathcal{M}$. □

Lemma 10 (Secrecy). *An adversary corrupting up to* t *parties in each layer learns no information about the secret* m*.*

Proof. Noticing that the secrecy property only makes sense when the dealer D is honest, we assume D is honest in the following proof. Therefore, we assume that the dealer D shares a valid 4-dimensional sharing ss_4 of m. We can notice that the shares held by the adversary are generated through a series of applications of the secrecy-preserving projection operation and the error-correcting operation. By the secrecy-preserving property of the projection operation, the shares held by the adversary after each projection reveal no information about m. The only potential leakage of information about m could occur in the broadcast messages. However, the broadcast messages consists of the shares held by P_i^1 for all i such that $\mathbf{R}[i] = 1$. By Lemma 1, any honest party P_i^1 satisfies $\mathbf{R}[i] = 0$, so only shares held by corrupted parties in $\mathcal{L}_1$ are broadcasted. Therefore the broadcast messages reveal no additional information about m. □

G Toy Example for Shamir-based VSS when $\mathbf{t = 1}$ and $\mathbf{n = 4}$

We illustrate our VSS construction with a toy example of the $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}$ protocol from Tab. 4. For this example, we assume, without loss of generality, that parties $\mathsf{P}_4^k \in \mathcal{L}_k$ are malicious. We analyze two cases separately: when the dealer is honest and when the dealer is malicious.

Case 1: Honest Dealer

- **Layer 2:** Each honest party P_j^2 obtains the shares shown in Figure 1(a). In Step 4, only the coordinates enclosed in the red boxes might be broadcast.
- **Layer 3:** Each honest party P_k^3 obtains the shares shown in Figure 1(b). Because the values within the yellow boxes are deterministic, they successfully reconstruct $\mathsf{F}^0(W, X, k, Z) = \mathsf{F}(W, X, k, Z)$ in Step 2. During the broadcast in Step 6, the only possible inequalities are $\mathsf{F}^0(i, j, k, Z) \neq \mathsf{F}^1(i, j, k, Z)$ within the blue boxes.
- **Public Computation 1:** It is only possible that $\mathbf{R}[4] := 1$.
- **Public Computation 2:** In Step 2, the parties are guaranteed to reconstruct $\hat{\mathsf{H}}(4, 0, Y, Z) = \mathsf{F}(4, 0, Y, Z)$.
- **Layer 5:** Each honest party P_m^5 utilizes the shares shown in Figure 1(c) for the final reconstruction. They can uniquely reconstruct the secret $\mathsf{F}(0, 0, 0, m)$.

Case 2: Malicious Dealer Consider a malicious dealer who samples two distinct polynomials, $\mathsf{T}(W, X, Y, Z)$ and $\mathsf{F}(W, X, Y, Z)$. The Dealer distributes the shares as follows: $\mathsf{T}(1, X, Y, Z)$ to P_1^1, and $\mathsf{F}(i, X, Y, Z)$ to P_i^1 for $i \in \{2, 3\}$. Note that the polynomials are deliberately crafted to satisfy $\mathsf{F}(1, 2, Y, Z) = \mathsf{T}(1, 2, Y, Z)$ and $\mathsf{F}(2, 1, Y, Z) = \mathsf{T}(2, 1, Y, Z)$. This condition aims to minimize the number of coordinates broadcast in $\mathcal{I}_j$ during step 4 in Layer 2, thereby concealing the malicious dealer's inconsistencies as much as possible.

- **Layer 1:** Given the conditions above, the malicious P_4^1 must send $\mathsf{F}(4, 2, k, Z)$ and $\mathsf{F}(4, 3, k, Z)$ to P_2^2 and P_3^2, respectively. Otherwise, an insufficient number of honest parties in Layer 3 will be able to recover $\mathsf{F}^0(W, X, k, Z)$ in Step 2, causing the protocol to abort at Step 10 of Public Computation 1 since the required public zero submatrix cannot be found. P_4^1 can send an arbitrary value to P_1^2; assume it sends $\mathsf{T}(1, 4, Y, Z)$.
- **Layer 2:** Each honest party P_j^2 obtains the shares shown in Figure 1(d). In Step 4, only the coordinates corresponding to the red values will be broadcast.
- **Layer 3:** At least two honest parties P_k^3 (assume $k \in \{1, 2\}$) obtain the shares shown in Figure 1(e). Otherwise, the number of reconstructed $\mathsf{F}^0(W, X, k, Z)$ polynomials in Step 3 would be insufficient, leading to an abort at Step 6 of Public Computation 1. P_3^3 obtains the shares shown in Figure 1(f), resulting in an empty polynomial: $\mathsf{F}^0(W, X, 3, Z) = \perp$.
  - *Step 6:* For $\mathcal{I}_3 = 1$, P_1^3 and P_2^3 broadcast $\mathsf{F}^0(1, 3, k, Z) = \mathsf{F}(1, 3, k, Z)$ and $\mathsf{F}^1(1, 3, k, Z) = \mathsf{T}(1, 3, k, Z)$ for $k \in \{1, 2\}$. P_3^3 broadcasts $\mathsf{F}^0(1, 3, 3, Z) = \perp$ and $\mathsf{F}^1(1, 3, 3, Z) = \mathsf{T}(1, 3, 3, Z)$. The malicious P_4^3 must broadcast $\mathsf{F}^0(1, 3, 4, Z) = \mathsf{F}(1, 3, 4, Z)$; otherwise, the protocol aborts at Step 6 of Public Computation 1. The same behavior applies for $\mathcal{I}_1 = 3$.
  - *Step 10 and 11:* The honest parties P_k^3 (for $k \in \{1, 2\}$) share $\mathsf{G}_{1,k}^0 = \mathsf{F}(1, 0, k, Z)$ to Layer 4. The malicious P_4^3 must share $\mathsf{G}_{1,4}^0 = \mathsf{F}(1, 0, 4, Z)$ to Layer 4; otherwise, the protocol aborts at Step 3 of Public Computation 2.
- **Public Computation 1:**
  - *Step 7:* Since $\mathsf{H}^0(1, 3, Y, Z) = \mathsf{F}(1, 3, Y, Z)$ and $\mathsf{H}^1(1, 3, Y, Z) = \mathsf{T}(1, 3, Y, Z)$, it is guaranteed in Step 7 that $\mathsf{H}^0(1, 3, Y, Z) \neq \mathsf{H}^1(1, 3, Y, Z)$, given the initial condition $\mathsf{T}(1, 2, Y, Z) = \mathsf{F}(1, 2, Y, Z)$ and the degree constraints of the polynomials. Additionally, $\mathsf{H}^0(3, 1, Y, Z) = \mathsf{H}^1(3, 1, Y, Z)$. As a result, $\mathbf{M}[1, j, k] := 1$ for each j, k.
  - *Step 9, 10 and 11:* Consider the case where the protocol does not abort at Step 10. During Step 4 of Layer 3, a 3×3 matrix satisfying $\mathcal{J}_k[i, j] = 0$ exists exclusively within the yellow boxes (as shown in Figure 1(e)) for at least one of P_1^3 and P_2^3. Since the polynomials are of degree 1, $\mathsf{T}(1, X, k, Z)$ may equal $\mathsf{F}(1, X, k, Z)$ for one party (e.g., P_1^3), but the degree constraint makes it impossible for them to be equal for both simultaneously. Therefore, the malicious P_4^3 is forced to broadcast identical $\mathcal{J}_k[i, j]$ values, ensuring the existence of a public, all-zero $3 \times 3 \times 3$ submatrix. As a result, $\mathbf{R}[1] := 1$.

– **Public Computation 2:** Assuming the protocol has not aborted, it successfully reconstructs $\hat{\sigma}_1^0(Z) = F(1,0,0,Z)$.
– **Layer 5:** After reconstructing the polynomial with respect to the V-variable, P_m^5 obtains the shares illustrated in Figure 1(g). Note the presence of the red $\mathsf{L}(4,0,3,m)$ value. This occurs because P_3^3's shares, $\mathsf{T}(4,1,k,Z) = \mathsf{L}(4,1,k,Z)$ and $\mathsf{F}(4,3,k,Z) = \mathsf{L}(4,3,k,Z)$ (marked in green in Figure 1(c)), cause it to share $\mathsf{G}^1_{4,0,3,Z} = \mathsf{L}(4,0,3,Z)$ to Layer 4. Despite this, the values within the yellow box provide sufficient information for P_m^5 to uniquely reconstruct the secret $\mathsf{F}(0,0,0,m)$.

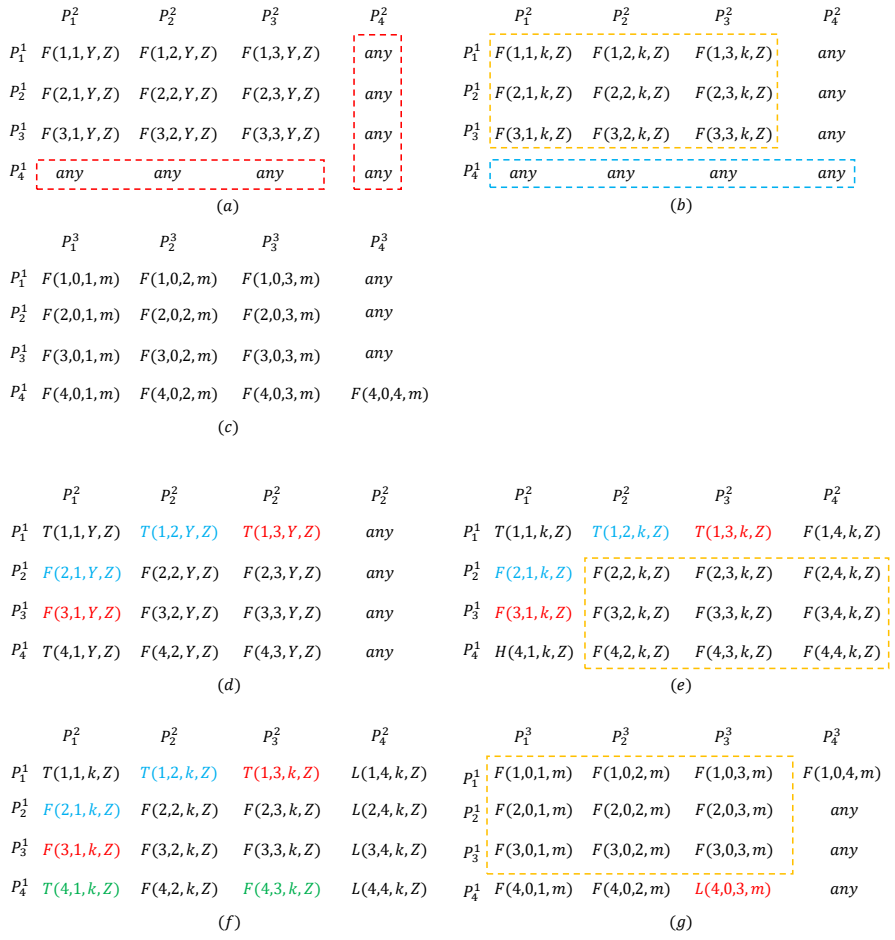


Fig. 1. Example of Shamir-based VSS.

H Construction of CNF-based VSS Protocol

We establish the following theorem regarding the security and efficiency of the CNF-based VSS protocol by applying the generic VSS construction from Tab. 3. Therefore, we omit the proof here.

Tab. 17: CNF-based Verifiable Secret Sharing Protocol $\Pi_{\mathsf{VSS}}^{\mathsf{CNF}}$

Setup:

Participants: Dealer $\mathsf{D} \in \mathcal{L}_0$, shareholders in $\mathcal{L}_4$ and internal computation layers $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$.

Secret Inputs: D has input $\mathsf{m} \in \mathcal{M}$.

Sub-protocols:

1. CNF-based robust secret sharing schemes $\mathsf{SS}_4^{\mathsf{CNF}}, \mathsf{SS}_3^{\mathsf{CNF}}, \mathsf{SS}_2^{\mathsf{CNF}}, \mathsf{SS}_1^{\mathsf{CNF}}$.
2. Majority rule $\mathsf{Maj}_T(\cdot)$ over index set T, which outputs the unique value with at least $\mathsf{t}+1$ identical copies, and outputs $\perp$ otherwise.

Notations: Let $\mathcal{S} = \{S \subseteq [\mathsf{n}] \mid |S| = \mathsf{n}-\mathsf{t}\} = (\mathcal{S}_1, \ldots, \mathcal{S}_{\mathsf{M}})$ be the family of all subsets of $[\mathsf{n}]$ of size $\mathsf{n}-\mathsf{t}$, where $\mathsf{M} = |\mathcal{S}| = \binom{\mathsf{n}}{\mathsf{t}}$.

Protocol Execution::

Dealer D:

1. Sample a random 4-dimensional CNF sharing $\{\mathsf{m}_{i,j,k,\ell}\}_{i,j,k,\ell\in[\mathsf{M}]}$ such that

$$\mathsf{m} = \sum_{i,j,k,\ell\in[\mathsf{M}]} \mathsf{m}_{i,j,k,\ell}.$$

2. **for** $i \in [\mathsf{M}]$ and $\mathsf{c}_1 \in \mathcal{S}_i$: **send** $\{\mathsf{m}_{i,j,k,\ell}\}_{j,k,\ell\in[\mathsf{M}]}$ **to** $\mathsf{P}_{\mathsf{c}_1}^1$ **in** $\mathcal{L}_1$.

Layer 1: For each $i \in [\mathsf{M}]$ and $\mathsf{c}_1 \in \mathcal{S}_i$, $\mathsf{P}_{\mathsf{c}_1}^1$ proceeds as follows:

1. **recv** $\{\mathsf{m}_{i,j,k,\ell}\}_{j,k,\ell\in[\mathsf{M}]}$ **from** D **in** $\mathcal{L}_0$.
2. **if** $\mathsf{ErrorDetect}(\{\mathsf{m}_{i,j,k,\ell}\}_{j,k,\ell\in[\mathsf{M}]}) \neq \emptyset$ **then BCast Complaint**.
3. **for** $j \in [\mathsf{M}]$ and $\mathsf{c}_2 \in \mathcal{S}_j$: **send** $\{\mathsf{m}_{i,j,k,\ell}\}_{k,\ell\in[\mathsf{M}]}$ **to** $\mathsf{P}_{\mathsf{c}_2}^2$ **in** $\mathcal{L}_2$.

Layer 2: For each $j \in [\mathsf{M}]$ and $\mathsf{c}_2 \in \mathcal{S}_j$, $\mathsf{P}_{\mathsf{c}_2}^2$ proceeds as follows:

1. **for** $i \in [\mathsf{M}]$ and $\mathsf{s}_1 \in \mathcal{S}_i$: **recv** $\{\mathsf{m}_{i,j,k,\ell}^{\mathsf{s}_1}\}_{k,\ell\in[\mathsf{M}]}$ **from** $\mathsf{P}_{\mathsf{s}_1}^1$ **in** $\mathcal{L}_1$.
2. **for** $i,k,\ell \in [\mathsf{M}]$: $\mathsf{m}_{i,j,k,\ell}^{\mathsf{c}_2} := \mathsf{Maj}_{\mathsf{s}_1\in\mathcal{S}_i}\left(\{\mathsf{m}_{i,j,k,\ell}^{\mathsf{s}_1}\}\right)$.
3. $\mathcal{I}_j^{\mathsf{c}_2} := \{\mathsf{s}_1 \mid \exists i,k,\ell \in [\mathsf{M}] : \mathsf{s}_1 \in \mathcal{S}_i \wedge (\mathsf{m}_{i,j,k,\ell}^{\mathsf{c}_2} = \perp \vee \mathsf{m}_{i,j,k,\ell}^{\mathsf{s}_1} \neq \mathsf{m}_{i,j,k,\ell}^{\mathsf{c}_2})\}$.
4. **BCast** $\mathcal{I}_j^{\mathsf{c}_2}$.
5. **for** $i,k,\ell \in [\mathsf{M}]$ and $\mathsf{s}_1 \in \mathcal{S}_i$: $\mathsf{m}_{i,j,k,\ell}^{\mathsf{s}_1,\mathsf{c}_2} := \begin{cases} \mathsf{m}_{i,j,k,\ell}^{\mathsf{c}_2} & \text{if } \mathsf{s}_1 \in \mathcal{I}_j^{\mathsf{c}_2} \\ \mathsf{m}_{i,j,k,\ell}^{\mathsf{s}_1} & \text{otherwise} \end{cases}$.
6. **for** $k \in [\mathsf{M}]$ and $\mathsf{c}_3 \in \mathcal{S}_k$: **send** $\{\mathsf{m}_{i,j,k,\ell}^{\mathsf{s}_1,\mathsf{c}_2}\}_{i,\ell\in[\mathsf{M}],\,\mathsf{s}_1\in\mathcal{S}_i}$ **to** $\mathsf{P}_{\mathsf{c}_3}^3$ **in** $\mathcal{L}_3$.

Public Computation 1:

1. **for** $j \in [\mathsf{M}]$: $\mathcal{I}_j := \bigcup_{\mathsf{c}_2\in\mathcal{S}_j} \mathcal{I}_j^{\mathsf{c}_2}$.
2. **if Complaint** is broadcast from P_i^1 **then for** $j \in [\mathsf{M}]$ with $i \in \mathcal{S}_j$: $\mathcal{I}_j := \mathcal{I}_j \cup \{i\}$.
3. **if** $\exists j \in [\mathsf{M}]$ such that $\forall i \in [\mathsf{M}]$, $\mathcal{I}_j \cap \mathcal{S}_i \neq \emptyset$ **then abort** .

Layer 3: For each $k \in [\mathsf{M}]$ and $\mathsf{c}_3 \in \mathcal{S}_k$, $\mathsf{P}_{\mathsf{c}_3}^3$ proceeds as follows:

1. **for** $j \in [\mathsf{M}]$ and $\mathsf{s}_2 \in \mathcal{S}_j$: **recv** $\{\mathsf{m}_{i,j,k,\ell}^{\mathsf{s}_1,\mathsf{s}_2}\}_{i,\ell\in[\mathsf{M}],\,\mathsf{s}_1\in\mathcal{S}_i}$ **from** $\mathsf{P}_{\mathsf{s}_2}^2$ **in** $\mathcal{L}_2$.
2. **for** $i,j,\ell \in [\mathsf{M}]$: $\mathsf{m}_{i,j,k,\ell}^{\mathsf{c}_3} := \mathsf{Maj}_{\mathsf{s}_1\in\mathcal{S}_i}\left(\mathsf{Maj}_{\mathsf{s}_2\in\mathcal{S}_j}\left(\{\mathsf{m}_{i,j,k,\ell}^{\mathsf{s}_1,\mathsf{s}_2}\}\right)\right)$.
3. **for** $i,j,\ell \in [\mathsf{M}]$: **if** $\mathcal{I}_j \cap \mathcal{S}_i \neq \emptyset$ **then BCast** $\mathsf{m}_{i,j,k,\ell}^{\mathsf{c}_3}$.
4. **for** $\ell \in [\mathsf{M}]$ and $\mathsf{c}_4 \in \mathcal{S}_\ell$: **send** $\{\mathsf{m}_{i,j,k,\ell}^{\mathsf{c}_3}\}_{i,j\in[\mathsf{M}]}$ **to** $\mathsf{P}_{\mathsf{c}_4}^4$ **in** $\mathcal{L}_4$.

Public Computation 2:

1. **for** $i,j,k,\ell \in [\mathsf{M}]$ **if** $\mathcal{I}_j \cap \mathcal{S}_i \neq \emptyset$ **then**
2. $\quad \dot{\mathsf{m}}_{i,j,k,\ell} := \mathsf{Maj}_{\mathsf{c}_3\in\mathcal{S}_k}\left(\{\mathsf{m}_{i,j,k,\ell}^{\mathsf{c}_3}\}\right)$.
3. $\quad$ **if** $\dot{\mathsf{m}}_{i,j,k,\ell} = \perp$ **then abort** .

Layer $\mathcal{L}_4$: For each $\ell \in [\mathsf{M}]$ and $\mathsf{c}_4 \in \mathcal{S}_\ell$, $\mathsf{P}_{\mathsf{c}_4}^4$ proceeds as follows:

1. **for** $k \in [\mathsf{M}]$ and $\mathsf{s}_3 \in \mathcal{S}_k$: **recv** $\{\mathsf{m}^{\mathsf{s}_3}_{i,j,k,\ell}\}_{i,j\in[\mathsf{M}]}$ **from** $\mathsf{P}^3_{\mathsf{s}_3}$ **in** $\mathcal{L}_3$.
2. $\mathsf{m}'_{i,j,k,\ell} := \begin{cases} \hat{\mathsf{m}}_{i,j,k,\ell} & \text{if } \mathcal{I}_j \cap \mathcal{S}_i \neq \emptyset \\ \mathsf{Maj}_{\mathsf{s}_3 \in \mathcal{S}_k}\left(\mathsf{m}^{\mathsf{s}_3}_{i,j,k,\ell}\right) & \text{otherwise} \end{cases}$.
3. **if** $\mathsf{m}'_{i,j,k,\ell} = \perp$ **then abort** .
4. Set the share m_ℓ as $\sum_{i,j,k\in[\mathsf{M}]} \mathsf{m}'_{i,j,k,\ell}$.

Theorem 5 (CNF-based VSS). *The protocol in Tab. 17 executed in parallel realizes $\mathcal{F}_{\mathsf{Par\text{-}VSS}}$ with perfect t-security for $\mathsf{t} < \mathsf{n}/3$ by consuming 4 layers, and by communicating $O(\mathsf{n}^3 \cdot \binom{\mathsf{n}}{\mathsf{t}}^4 \log|\mathbb{G}|)$ over the point-to-point channels and $O(\mathsf{n} \cdot \binom{\mathsf{n}}{\mathsf{t}}^4 \log|\mathbb{G}|)$ over the broadcast channel for each secret.*

I Proof and Complexity Analysis of Theorem 3

Proof. We prove that above theorem by specifying a simulator Sim that simulates for all environments $\mathcal{E}$ and all adversaries $\mathcal{A}$ corrupting at most t parties in each layer. The simulator Sim operates as follows:

- **Environment Interaction.** Sim gets all honest shares from the environment $\mathcal{E}$ and forwards them to the ideal functionality $\mathcal{F}_{\mathsf{Mult}}$. As soon as $\mathcal{F}_{\mathsf{Mult}}$ outputs the result, Sim receives it and forwards it to $\mathcal{E}$. Noticing that to launch the protocol, the environment $\mathcal{E}$ needs to provide the inputs of all honest parties, from which Sim can fully recover all secret shares with robust secret sharing. Moreover, the completeness established in Lemma 12 ensures that the output is valid shares of the multiplication result for all honest parties. Thus, the view simulated by *Sim* for $\mathcal{E}$ is identical in both real and ideal worlds.
- **Corrupted Parties Interaction.** Sim internally simulates all honest parties and interacts with the adversary $\mathcal{A}$ on their behalf, then Sim forwards all messages sent by $\mathcal{A}$ to outside the protocol to the environment $\mathcal{E}$. By the secrecy property of $\mathbf{\Pi}^{\mathsf{Shamir}}_{\mathsf{Mul}}$ given in Lemma 13, Sim can perfectly simulate the view of $\mathcal{A}$ in the real execution. This is because Sim can extract the inputs of all corrupted parties from $\mathcal{A}$ and simulate the honest parties' messages using random values, which are indistinguishable from the real execution due to the secrecy property.

By the properties of robust secret sharing, honest future resharing, honest future messaging, and verifiable secret sharing, we conclude that the view of the adversary $\mathcal{A}$ in the simulated execution is indistinguishable from that in the real execution. Therefore, the protocol securely realizes the ideal functionality $\mathcal{F}_{\mathsf{Mult}}$ with perfect security against up to $\mathsf{t} < \mathsf{n}/3$ corruptions. □

Before proving Theorem 3, we first establish some useful lemmas. We begin by showing that if an honest party in $\mathcal{L}_0$ shares valid inputs, then all honest parties in $\mathcal{L}_5$ will receive consistent shares derived from bivariate polynomials.

Lemma 11. *If a party P^0_i in $\mathcal{L}_0$ is honest, then for any honest party P^5_j in $\mathcal{L}_5$, the values $[\![a]\!][i,j]$ and $[\![b]\!][i,j]$ it computes are consistent with bivariate polynomials $\mathsf{F}_a(X,Y)$ and $\mathsf{F}_b(X,Y)$ of degree t in each variable.*

Proof. Consider an honest party P_i^0 in $\mathcal{L}_0$. It applies the Lift function to its share $[\![a]\!][i]$ and random shares $\{[\![r_l]\!][i]\}_{l\in[\mathsf{t}]}$ to define a degree-t polynomial $\mathsf{F}_a(i, Y)$. The shares $\{[\![a]\!][i]\}_{i\in[\mathsf{n}]}$ and $\{[\![r_l]\!][i]\}_{i\in[\mathsf{n}]}$ for each l lie on degree-t polynomials in i. Consequently, the set of polynomials $\{\mathsf{F}_a(i, Y)\}_{i\in[\mathsf{n}]}$ defines a bivariate polynomial $\mathsf{F}_a(X, Y)$ of degree t in both variables.

P_i^0 then sends the evaluation $\mathsf{F}_a(i, j)$ to each party P_j^5 in $\mathcal{L}_5$ using the honest future messaging protocol $\mathbf{\Pi}_{\mathsf{HFM}}$. Since at most t parties in $\mathcal{L}_0$ are corrupt, the properties of $\mathbf{\Pi}_{\mathsf{HFM}}$ guarantee that each honest P_j^5 receives at least $\mathsf{n} - \mathsf{t} > 2\mathsf{t}$ correct values $\{\mathsf{F}_a(i, j)\}_{i\in H}$ where H is the set of honest parties in $\mathcal{L}_0$.

Using Berlekamp-Welch decoding on the received values $\{\mathsf{F}_a(i, j)\}_{i\in[\mathsf{n}]}$, P_j^5 can uniquely reconstruct the degree-t polynomial $\hat{\mathsf{F}}_a(X, j) = \mathsf{F}_a(X, j)$. Evaluating this polynomial at $X = i$ yields $\hat{\mathsf{F}}_a(i, j) = \mathsf{F}_a(i, j) = [\![a]\!][i, j]$. An analogous argument holds for $[\![b]\!][i, j]$. □

Lemma 12 (Completeness). *If there are at most* t *parties are corrupted in each layer, then for each* $v \in [\mathsf{n}]$*, the party* P_v^8 *in* $\mathcal{L}_8$ *outputs a share* $[\![c]\!][v]$ *such that the set of shares* $\{[\![c]\!][v]\}_{v\in[\mathsf{n}]}$ *constitutes a valid* t*-degree sharing of the product* $c = ab$*.*

Proof. By the strong committing property of the $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}$ protocol, at the beginning of $\mathcal{L}_5$, each party P_i^0 is committed to three polynomials $f_i(x), g_i(x), h_i(x)$, where $h_i(x) = f_i(x)g_i(x)$ committing all coefficients. We first prove that i is not discarded during the public computation phase if and only if, for all honest P_j^5 in $\mathcal{L}_5$, the messages they received from honest P_i^0 satisfy the following two conditions:

1. $[\![c]\!][i] = [\![a]\!][i] \cdot [\![b]\!][i]$,
2. the committed polynomials $\{f_i(x), g_i(x), h_i(x)\}$ verifies that $f_i(x) \cdot g_i(x) = h_i(x)$.

First, we establish that for any honest party P_i^0, no honest party will complain, and thus i will not be discarded. By Lemma 11, for any honest P_j^5, the values $\hat{\mathsf{F}}_a(i, j)$ and $\hat{\mathsf{F}}_b(i, j)$ it reconstructs are consistent with all other honest parties. Furthermore, the properties of $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}$ ensure that the shares of the coefficients of $f_i(x)$, $g_i(x)$, and $h_i(x)$ held by honest parties in $\mathcal{L}_5$ from honest parties in $\mathcal{L}_1$ are consistent. Since Shamir's secret sharing is t-robust, and there are at least $2\mathsf{t} + 1$ honest parties in $\mathcal{L}_5$, any honest party P_k^6 in $\mathcal{L}_6$ will correctly reconstruct $\hat{f}_i(k) = f_i(k)$, $\hat{g}_i(k) = g_i(k)$, and $\hat{h}_i(k) = h_i(k)$. Because P_i^0 is honest, it sets $h_i(x) = f_i(x)g_i(x)$, which implies $\hat{h}_i(k) = \hat{f}_i(k)\hat{g}_i(k)$. Thus, no honest party P_k^6 will broadcast a complaint, and i will not be discarded.

Next, we show that if a dishonest party P_i^0 violates either of the two conditions, it will be discarded.

1. Suppose P_i^0 is dishonest and sets the constant term of $h_i(x)$ to $[\![c']\!][i] \neq [\![a]\!][i] \cdot [\![b]\!][i]$. Let $h_i(x)$ be the polynomial shared by P_i^0 and $h_i^*(x) = f_i(x)g_i(x)$ be the correctly computed polynomial. Since $h_i(x)$ and $h_i^*(x)$ are distinct polynomials of degree at most $2\mathsf{t}$, they can agree on at most $2\mathsf{t}$ points. As there

are at least $2\mathsf{t}+1$ honest parties in $\mathcal{L}_6$, there must be at least one honest party P_k^6 for whom $h_i(k) \neq h_i^*(k)$. This party will correctly reconstruct $\hat{f}_i(k) = f_i(k)$ and $\hat{g}_i(k) = g_i(k)$, but will find that $\hat{f}_i(k)\hat{g}_i(k) = h_i^*(k) \neq h_i(k) = \hat{h}_i(k)$. Consequently, P_k^6 will broadcast a complaint $(\textbf{Complaint}, i, k)$.
During the adjudication in $\mathcal{L}_7$, all honest parties broadcast their shares related to the complaint. Since there are more than $2\mathsf{t}$ honest parties in $\mathcal{L}_5$ and $\mathcal{L}_7$, the values $f_i(k)$, $g_i(k)$, and $h_i(k)$ will be publicly and correctly reconstructed. The public verification will confirm that $f_i(k)g_i(k) \neq h_i(k)$, causing i to be marked as $\mathsf{discarded}$.

2. Suppose P_i^0 sets the constant term correctly but chooses other coefficients for $h_i(x)$ such that $h_i(x) \neq f_i(x)g_i(x)$. The argument is analogous to the case above. The polynomials $h_i(x)$ and $f_i(x)g_i(x)$ are distinct, so at least one honest party P_k^4 will detect the discrepancy and issue a complaint, leading to P_i^0 being discarded.

In summary, a party P_i^0 is not discarded if and only if it correctly follows the protocol in $\mathcal{L}_0$. This ensures that for any non-discarded i, the shares $\{[\![c]\!][i,j]\}_{j\in[\mathsf{n}]}$ held by parties in $\mathcal{L}_5$ form a valid sharing of $[\![c]\!][i] = [\![a]\!][i][\![b]\!][i]$. By the correctness of $\mathbf{\Pi}_{\mathsf{FRS}}$, the shares $\{[\![c]\!][i,v]\}_{v\in[\mathsf{n}]}$ received by parties in $\mathcal{L}_8$ also constitute a valid sharing of $[\![c]\!][i]$. Finally, the output computation in $\mathcal{L}_8$ correctly combines these shares to produce a valid sharing of the product $c = ab$. □

Lemma 13 (Secrecy). *The protocol $\mathbf{\Pi}_{\mathsf{Mul}}^{\mathsf{Shamir}}$ is perfectly secure against an adversary corrupting at most t parties in each layer.*

Proof. We argue that the view of an adversary corrupting up to t parties in each layer can be perfectly simulated. The adversary's view consists of the internal states of corrupted parties and all messages they receive. These messages are either from sub-protocols $(\mathbf{\Pi}_{\mathsf{RG}}, \mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}, \mathbf{\Pi}_{\mathsf{HFM}}, \mathbf{\Pi}_{\mathsf{HFS}}, \mathbf{\Pi}_{\mathsf{FRS}})$ or are broadcast messages during the complaint phase.

The security of the sub-protocols guarantees that the messages received by corrupted parties can be simulated. Specifically, the messages are either random values or are shares that are computationally indistinguishable from random values from the perspective of an adversary who does not possess a sufficient number of shares.

The only other messages are public broadcasts resulting from complaints. A complaint $(\textbf{Complaint}, i, k)$ is broadcast by a party P_k^6 in $\mathcal{L}_6$ only if the check $\hat{f}_i(k) \cdot \hat{g}_i(k) \neq \hat{h}_i(k)$ fails. As established in the proof of Lemma 12, this can only happen if P_i^0 is dishonest. If P_i^0 is dishonest, the adversary controls it and is aware of the malformed data it sent. The subsequent adjudication in $\mathcal{L}_7$ involves parties broadcasting their shares. Since P_i^0 is corrupt, the adversary already possesses the information that leads to the complaint and the subsequent public reconstruction. Therefore, all broadcast messages can be simulated.

Since all messages received by the adversary can be perfectly simulated based on the shares held by the corrupted parties and publicly available information, the adversary learns no additional information about the secrets a, b, or their product ab. This establishes the perfect security of the protocol. □

Complexity Analysis We analyze the communication complexity of the protocol layer by layer.

- **Layer 0:** Each of the n parties P_i^0 performs the following actions:
  - Invokes $\mathbf{\Pi}_{\mathsf{RG}}$ once to obtain $2\mathsf{t}$ random shares, resulting in a total communication of $O(\mathsf{n}^5)$.
  - Invokes $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}$ $4\mathsf{t}+1$ times to share the coefficients of the polynomials $f_i(x)$, $g_i(x)$, and $h_i(x)$. Since each $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}$ invocation costs $O(\mathsf{n}^4)$ and $\mathsf{t}=\Theta(\mathsf{n})$, this step amounts to $\mathsf{n}\cdot O(\mathsf{n})\cdot O(\mathsf{n}^3)=O(\mathsf{n}^6)$ communication.
  - Invokes $\mathbf{\Pi}_{\mathsf{HFM}}$ $2\mathsf{n}$ times to send evaluations to parties in $\mathcal{L}_5$. Each invocation costs $O(\mathsf{n}^4)$, leading to a total of $\mathsf{n}\cdot 2\mathsf{n}\cdot O(\mathsf{n}^4)=O(\mathsf{n}^6)$ communication.
- **Layer 5:** Each of the n parties P_j^5 performs the following:
  - Invokes $\mathbf{\Pi}_{\mathsf{HFS}}$ $3\mathsf{n}^2$ times to share evaluations with $\mathcal{L}_7$. Each invocation costs $O(\mathsf{n}^2)$, resulting in a total of $\mathsf{n}\cdot 3\mathsf{n}^2\cdot O(\mathsf{n}^2)=O(\mathsf{n}^5)$ communication.
  - Invokes $\mathbf{\Pi}_{\mathsf{FRS}}$ n times to reshare $\{[\![c]\!][i,j]\}_{i\in[\mathsf{n}]}$ with $\mathcal{L}_8$. This contributes another $\mathsf{n}\cdot\mathsf{n}\cdot O(\mathsf{n}^4)=O(\mathsf{n}^6)$ to the total communication.
- **Layers 6 and 7:** The broadcasts for complaints in these layers are bounded. In the worst case, all t malicious parties raise complaints against all n parties, leading to $O(\mathsf{n}^2)$ broadcasts. The subsequent public reconstruction involves broadcasting shares, which costs $O(\mathsf{n}^4)$ broadcast communication in total.

The dominant cost over the point-to-point channels is $O(\mathsf{n}^6)$, stemming from the $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}$, $\mathbf{\Pi}_{\mathsf{HFS}}$, and $\mathbf{\Pi}_{\mathsf{FRS}}$ sub-protocols. Therefore, the overall communication complexity of the Shamir's multiplication protocol in Tab. 5 is $O(\mathsf{n}^6)$ over the point-to-point channels and $O(\mathsf{n}^6)$ over the broadcast channels.

J Secure Multiplication and MPC for CNF-based secret sharing

Details of CNF-based Multiplication are given in Tab. 18 and we omit the proof.

Theorem 6. *The protocol in Tab. 18 realizes the ideal functionality $\mathcal{F}_{\mathsf{Mult}}$ in the $\mathcal{F}_{\mathsf{VSS}}^{(\mathsf{t},\mathsf{N})}$-hybrid model as defined in Tab. 11 with perfect t-security for $\mathsf{t}<\mathsf{n}/3$ by consuming 7 layers, by communicating $2^{O(\mathsf{n})}$ ring elements over the point-to-point channels and by communicating $2^{O(\mathsf{n})}$ ring elements over the broadcast channels.*

Tab. 18: CNF-based Multiplication $\Pi_{\mathsf{Mul}}^{\mathsf{CNF}}$

Setup:

Public Parameters: Input parties $\mathsf{P}_1^0,\ldots,\mathsf{P}_n^0\in\mathcal{L}_0$, output parties $\mathsf{P}_1^7,\ldots,\mathsf{P}_n^7\in\mathcal{L}_7$.
Secret Input: Each $\mathsf{P}_i^0\in\mathcal{L}_0$ input $[\![a]\!][i]=\{a_l\}_{i\in\mathcal{S}_l,l\in\mathsf{M}}$ and $[\![b]\!][i]=\{b_l\}_{i\in\mathcal{S}_l,l\in\mathsf{M}}$.
Sub-protocols:
- Protocol $\mathbf{\Pi}_{\mathsf{HFM}},\mathbf{\Pi}_{\mathsf{HFS}},\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{CNF}}$

Protocol Execution:

Layer 0: For each $i,j\in[\mathsf{M}]$, define a set $I_{i,j}=\mathcal{S}_i\cap\mathcal{S}_j$, for each $k\in\mathcal{S}_i\cup\mathcal{S}_j$, P_k^0 proceeds as follows

1. **if** $k\in I_{i,j}, k=min(I_{i,j})$:

2. sample $(\delta_1^{i,j}, \delta_2^{i,j}, \cdots, \delta_{\mathsf{M}}^{i,j})$ such that $a_i b_j = \sum_{u=1}^{\mathsf{M}} \delta_u^{i,j}$
3. share each $\delta_u^{i,j}, u \in [\mathsf{M}]$ with $\mathcal{L}_5$ via $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{CNF}}$.
4. **elseif** $k \in I_{i,j}, k \neq min(I_{i,j})$
5. share $(a_i b_j)^k$ with $\mathcal{L}_5$ via $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{CNF}}$. ($(a_i b_j)^k$ means $a_i b_j$ hold by P_k^0.)
6. **if** $k \in \mathcal{S}_i$
7. share a_i^k with $\mathcal{L}_6$ via $\mathbf{\Pi}_{\mathsf{HFS}}$. ($a_i^k$ means a_i hold by P_k^0.)
8. **if** $k \in \mathcal{S}_j$
9. share b_j^k with $\mathcal{L}_6$ via $\mathbf{\Pi}_{\mathsf{HFS}}$.

Layer 4: For each $i, j \in [\mathsf{M}]$, each $v \in [\mathsf{M}], \ell \in \mathcal{S}_v$, P_ℓ^4 proceeds as follows

1. **recv** $\{(a_i b_j)_v^k\}_{k \in I_{i,j}, k \neq min(I_{i,j})}, \{\delta_{u,v}^{i,j}\}_{u \in [\mathsf{M}]}$ **from** $\mathcal{L}_0$.
2. **for** each $k \in I_{i,j}, k \neq min(I_{i,j})$
3. **BCast** $((a_i b_j)_v^k - \sum_{u=1}^{\mathsf{M}} \delta_{u,v}^{i,j})^\ell$. ($(a_i b_j)_v^k$ means the v-th share of $(a_i b_j)^k$.)
4. **for** each $u \in [\mathsf{M}]$
5. **send** $(\delta_{u,v}^{i,j})^\ell$ **to** each $\mathsf{P}_m^7, m \in \mathcal{S}_u$ via $\mathbf{\Pi}_{\mathsf{HFM}}$.

Layer 5: For each $i, j \in [\mathsf{M}]$, each $l \in [\mathsf{M}], i' \in \mathcal{S}_l$, $\mathsf{P}_{i'}^5$ proceeds as follows

1. **recv** $\{a_{i,l}^k\}_{k \in \mathcal{S}_i}, \{b_{j,l}^k\}_{k \in \mathcal{S}_j}$ **from** $\mathcal{L}_0$.
2. **for** each $k \in I_{i,j}, k \neq min(I_{i,j})$:
3. **for** each $v \in [\mathsf{M}]$: $(a_i b_j)_v^k - \sum_{u=1}^{\mathsf{M}} \delta_{u,v}^{i,j} = \mathsf{Maj}(\{((a_i b_j)_v^k - \sum_{u=1}^{\mathsf{M}} \delta_{u,v}^{i,j})^\ell\}_{\ell \in \mathcal{S}_v})$
4. $\lambda^k := \sum_{v \in [\mathsf{M}]} (a_i b_j)^k - \sum_{u=1}^{\mathsf{M}} \delta_{u,v}^{i,j}$.
5. **if** $\lambda^k \neq 0$:
6. **BCast** (**Complaint**, $i, j, \{a_{i,l}^{k,i'}\}_{k \in \mathcal{S}_i}, \{b_{j,l}^{k',i'}\}_{k' \in \mathcal{S}_j}$). ($a_{i,l}^{k,i'}$ means $a_{i,l}^k$ hold by $\mathsf{P}_{i'}^4$)
7. **Break**

Layer 6: For each $u \in [\mathsf{M}], m \in \mathcal{S}_u$, P_m^6 proceeds as follows

1. **recv** $\{(\delta_{u,v}^{i,j})^\ell\}_{v \in [\mathsf{M}], \ell \in \mathcal{S}_v}$ **from** $\mathcal{L}_4$.
2. **for** each $i, j \in [\mathsf{M}]$:
3. **if** there are at least $\mathsf{n} - \mathsf{t}$ complaints about i, j
4. **for** $k \in \mathcal{S}_i, k' \in \mathcal{S}_j$:
5. **for** $l \in [\mathsf{M}]$:$a_{i,l}^k = \mathsf{Maj}(\{a_{i,l}^{k,i'}\}_{i' \in \mathcal{S}_l}), b_{j,l}^{k'} = \mathsf{Maj}(\{b_{j,l}^{k',i'}\}_{i' \in \mathcal{S}_l})$
6. $a_i^k = \sum_{l \in [\mathsf{M}]} a_{i,l}^k, b_j^{k'} = \sum_{l \in [\mathsf{M}]} b_{j,l}^{k'}$.
7. Set $a_i = \mathsf{Maj}(\{a_i^k\}_{k \in \mathsf{Set}_i}), b_j = \mathsf{Maj}(\{b_j^{k'}\}_{k' \in \mathsf{Set}_j})$.
8. **if** $u = 1$: set $\delta_u^{i,j} = a_i b_j$
9. **else** set $\delta_u^{i,j} = 0$.
10. **for** each $i, j \in [\mathsf{M}]$
11. **if** there are less than $\mathsf{t} + 1$ complaints about i, j
12. **for** each $v \in [\mathsf{M}]$: $\delta_{u,v}^{i,j} = \mathsf{Maj}(\{\delta_{u,v}^{i,j,\ell}\}_{\ell \in \mathcal{S}_v})$
13. $\delta_u^{i,j} = \sum_{v=1}^{\mathsf{M}} \delta_{u,v}^{i,j}$.
14. Compute $\sum \sum_{i,j \in [\mathsf{M}]} \delta_u^{i,j}$ and set the result as its share of ab.

Details of CNF-based Multiplication Gate are given in Tab. 19.

Tab. 19: CNF-based Multiplication Gate $\mathbf{\Pi}_{\mathsf{Mul\text{-}Gate}}^{\mathsf{CNF}}$

Setup:

Public Parameters: Input parties $\mathsf{P}_1^k, \ldots, \mathsf{P}_n^k \in \mathcal{L}_k (k \geq 11)$, output parties $\mathsf{P}_1^{k+1}, \ldots, \mathsf{P}_n^{k+1} \in \mathcal{L}_{k+1}$.
Secret Inputs: Each P_i^k inputs $[\![x]\!][i]$ and $[\![y]\!][i]$.
Sub-protocols:
– Protocol $\mathbf{\Pi}_{\mathsf{FRS}}, \mathbf{\Pi}_{\mathsf{RG}}, \mathbf{\Pi}_{\mathsf{Mul}}^{\mathsf{CNF}}$.

Triple Generation:

1. $\mathcal{L}_{k-9}$ invokes $\mathbf{\Pi}_{\mathsf{RG}}$, and each P_l^{k-5} get random shares $[\![a]\!][l]$ and $[\![b]\!][l]$.
2. $\mathcal{L}_{k-5}$ invokes $\mathbf{\Pi}_{\mathsf{Mul}}^{\mathsf{CNF}}$ with the shares of a and b as input, and each P_l^{k+1} gets $[\![ab]\!][l]$.
3. $\mathcal{L}_{k-5}$ invokes $\mathbf{\Pi}_{\mathsf{FRS}}$ to reshare a and b to $\mathcal{L}_{k+1}$, thus $\mathcal{L}_k$ and $\mathcal{L}_{k+1}$ can obtain refresh shares of a and b.

Protocol Execution:

Layer k(Gate Input Layer): For each $i \in [\mathsf{M}], j \in \mathcal{S}_i$, P_i^k proceeds as follows:

1. Broadcast $(x-a)_i^j$ and $(y-b)_i^j$.

Layer $k+1$(Gate Output Layer): For each $k \in [\mathsf{M}], \ell \in \mathcal{S}_k$, P_ℓ^{k+1} proceeds as follows:

1. **for** each $i \in [\mathsf{M}]$: $(x-a)_i = \mathsf{Maj}(\{(x-a)_i^j\}_{j\in\mathcal{S}_i}), (y-b)_i = \mathsf{Maj}(\{(y-b)_i^j\}_{j\in\mathcal{S}_i})$
2. $\quad d = \sum_{i\in[\mathsf{M}]}(x-a)_i, e = \sum_{i\in[\mathsf{M}]}(y-b)_i$
3. Set $(xy)_j = d \cdot b_j + e \cdot a_j + (ab)_j + \frac{de}{\mathsf{M}}$.

Details of CNF-based Layered MPC Protocol are given in Tab. 20.

Theorem 7 (CNF-based MPC). *Let f be an n-party functionality computed by a general arithmetic circuit C over a finite field $\mathbb{F}$, with D levels and M gates. Then, for any $\mathsf{t} < \mathsf{n}/3$, there is an $(\mathsf{n}, \mathsf{t}, D+10)$-layered MPC protocol for f in which the communication is $M \cdot 2^{O(\mathsf{n})}$ field elements over the point-to-point channels and $M \cdot 2^{O(\mathsf{n})}$ field elements over the broadcast channels.*

Tab. 20: CNF-based Layered MPC Protocol

Setup:

Public Parameters: A arithmetic circuit C over $\mathbb{F}$ with D levels that computes f. Output layer is $d = D + 10$.
Secret Input: Each input client $\mathsf{P}_i^0, i \in [\mathsf{n}]$ inputs x_i.
Outputs: Each output client receives $f(x_1, \ldots, x_n)$.
Sub-protocols:

- Protocol $\mathbf{\Pi}_{\mathsf{FRS}}, \mathbf{\Pi}_{\mathsf{Mul}}^{\mathsf{CNF}}, \mathbf{\Pi}_{\mathsf{Mul\text{-}Gate}}^{\mathsf{CNF}}, \mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{CNF}}$.

Protocol Execution:

Input Sharing Phase:

- Each P_i^0 sample $r_k^i \in \mathcal{M}$ for each $k \in [\mathsf{nM}]$, and invokes $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{CNF}}$ with input r_k^i. In $\mathcal{L}_3$, each party obtain the share of r_k^i, noted as $[\![r_k^i]\!]$, then computes $\sum_{i\in[\mathsf{n}]}[\![r_k^i]\!]$ for each $k \in [\mathsf{nM}]$ as the random shares.
- Each P_i^0 invokes $\mathbf{\Pi}_{\mathsf{VSS}}^{\mathsf{Shamir}}$ to share its inputs x_i to $\mathcal{L}_3$. Then, $\mathcal{L}_3$ invokes $\mathbf{\Pi}_{\mathsf{FRS}}$ to deliver the share to the **Gate Input Layer** of its input wire.

Circuit Evaluation Phase (Starting at $\mathcal{L}_7$):

- **Mutiplication Gate.** For each $i \in [\mathsf{n}]$, P_i^k inputs $[\![x]\!][i]$ and $[\![y]\!][i]$ at **Gate Input Layer**($\mathcal{L}_k$) and invokes $\mathbf{\Pi}_{\mathsf{Mul\text{-}Gate}}^{\mathsf{CNF}}$, each P_j^{k+1} outputs $[\![xy]\!][j]$ at **Gate Output Layer**($\mathcal{L}_{k+1}$).
- **Addition Gate.** For each $i \in [\mathsf{n}]$, P_i^k inputs $[\![x]\!][i]$ and $[\![y]\!][i]$ at **Gate Input Layer**($\mathcal{L}_k$) and each P_j^k output $[\![(x+y)]\!][j] = [\![x]\!][j] + [\![y]\!][j]$ at **Gate Output Layer**($\mathcal{L}_k$).
- If the current gate's **Gate Output Layer** does not match its successor's **Gate Input Layer**, invoke $\mathbf{\Pi}_{\mathsf{FRS}}$ to reshare input share to its successor's **Gate Input Layer**.

Output Reconstruction Phase:

- For each $i \in [\mathsf{M}], j \in [\mathsf{n}]$, P_j^{D+9} in $\mathcal{L}_{D+9}$ reveals its share y_i on the output wire of the output gate to each output clients in $\mathcal{L}_{D+10}$.
- Each output client in $\mathcal{L}_{D+10}$ proceeds as follows:
- $\quad$ **for** each $i \in [\mathsf{M}]$: compute $y_i = \mathsf{Maj}(\{y_i^j\}_{j\in\mathcal{S}_i})$
- $\quad y = \sum_{i\in[\mathsf{n}]} y_i$.
- Set y as the output of the protocol.