

Advancing the Communication Complexity Landscape of Perfectly Secure Multiparty Computation

A THESIS
SUBMITTED FOR THE DEGREE OF
Doctor of Philosophy
IN THE
Faculty of Engineering

BY
Patil Shravani Mahesh



Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

July, 2024

Declaration of Originality

I, **Patil Shravani Mahesh**, with SR No. **04-04-00-10-12-19-1-17037** hereby declare that the material presented in the thesis titled

Advancing the Communication Complexity Landscape of Perfectly Secure Multiparty Computation

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2019-2024**.

With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

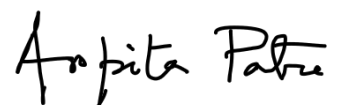
Date: Thursday 4th July, 2024



Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Arpita Patra



Advisor Signature

© Patil Shravani Mahesh
July, 2024
All rights reserved

DEDICATED TO

Pappa and Mamma

Acknowledgements

I would like to express my deepest gratitude to my advisor, Dr. Arpita Patra, for her unwavering support and invaluable guidance throughout this journey. Her critical insights have constantly pushed me to strive for excellence. Working with her has been an invaluable learning experience, teaching me the essential qualities and attributes of a dedicated researcher. Her extensive knowledge and passion for the field have been a constant source of inspiration, motivating me to continue expanding my horizons.

I would not have been at IISc if it had not been for my master's advisor, Dr. Purushothama B R, who believed in me more than I did. His work ethic and passion for teaching are qualities that I sincerely admire and look up to. I am grateful to Purushothama sir for his support and the invaluable life lessons that he has taught me.

During the course of my Ph.D., I am fortunate to have had the opportunity to work with phenomenal researchers. I would like to thank Dr. Ittai Abraham for his persistence in summarizing the high-level idea of our projects and his boundless optimism, which kept the project going even when we felt we were stuck. My gratitude goes to Prof. Gilad Asharov for his invaluable feedback on writing papers and for his attention to the details of our projects, which taught me a lot. Working with them has been an invaluable addition to my research journey, and I am grateful to have had this opportunity. I am thankful to Ajith Suresh and Nishat Koti for their guidance and genuine feedback. Working with them on one of my first projects helped me start out with research with ease. It has been a wonderful experience to have shared my Ph.D. journey with Protik Paul. Discussions with him have been an enriching experience and I am grateful to have had him as a collaborator and a friend. I also extend my gratitude to Varsha Bhat Kukkala, Aditya Hegde, Anirudh Chandramouli, and Ananya Appan. Working with them has been a great pleasure and a valuable learning experience. I am thankful to Divya Ravi for her motivation and support during my initial days at the institute.

The CrIS Lab has been a place where learning has been enjoyable, and I have had the privilege of interacting with many amazing people. I will always cherish the time spent with

Acknowledgements

Protik, Abhishek, Moumita, Masavir, Bhavish, Banashri, Shyam, Riddhiman, and Siddharth. Be it discussing some research ideas or playing board games, every lab member has contributed immensely to this research journey, and I am truly grateful to have shared the lab space with all of them.

I would like to express my sincere gratitude to the CSA office staff – Kushael ma’am, Padma ma’am, Nishita ma’am, Meenakshi ma’am, Aravind, Shubha ma’am for their invaluable assistance with administrative procedures and patient resolution of my queries over the years. I would also like to acknowledge the CSA technical staff, particularly Akshaynath, for his dedication and patience in addressing queries related to lab equipment. I am also thankful to the CSA security guards, especially Sudesh bhaiyya for being extremely supportive. I express my sincere gratitude to the security staff of IISc for going out of their way to help us with the search and rescue of many animals on campus with unmatched dedication. During this period of five years, many people outside of IISc ensured that I felt at home in Bangalore. I am grateful to Bel Road Amma, Ashraf, Cherry, and Babu Bhaiyya for their generosity and care.

I have been fortunate to have friends who made my stay at IISc enjoyable and truly memorable. I cherish the time spent with Faizan, Kashif, Zeeshan, Javed, Masavir, Ayush, Mayur, and Sagar. Be it playing badminton, exploring various cafes and restaurants, or having a debate on which variety of mangoes tastes the best, their company has made each of these activities special. I am sincerely thankful to Masavir for being a genuine well-wisher who constantly prayed and cheered for me. I am also thankful to Nikhil Gupta, for his wonderful company and generosity in cooking delicious meals for us. I would like to express my gratitude to Suraj for his company and friendship. A special acknowledgment also goes to my friends from NITG, Kanna and Perkin, for always being available for a conversation and giving me company virtually throughout this period. I am deeply grateful to my friends back home – Apeksha, Atish, and Sandeep – who have always believed in me and stood by me. Their relentless effort to keep in touch is something I sincerely keep close to my heart.

Growing up in Goa has been a blessing, with a close community that feels like a big family. Every person associated with my father’s workshop has had a significant role in shaping my personality, and I am truly grateful to them. I am forever indebted to Dr. Girish Wagle and Dada Mama for their immense care. I extend my deepest gratitude to Vaibhav Tamba, Rajesh Pai, Dennis and Melba D’Sa, Efraim D’Sa, and Joseph Monteiro for their support and encouragement. I am thankful to Golda Aunty, Emmanuel Meneses, Sasha Pinto, and Adi for always being there for my parents. Your support has brought me comfort, and I am truly grateful to have you all. I am thankful to Vaibhav Tamba, Kris Furtado, Keith Furtado, and

Acknowledgements

Patrick uncle for believing in me and ensuring that I continue to pursue my hobbies. I also extend my gratitude to Dipesh Kunkolienkar, Meghana Kunkolienkar, Priya Chimulkar, Girish Chimulkar, Aaji, Mitul, and Kunda aunty for their unwavering love and support.

Finally, I would like to express my deepest gratitude to my family. To all my furry and feathered companions - Chini, Chani, Popu, Leo, Ginny, Tiffy, Topsy, Brownie, Sifa, Sooby, Coffee, Blacky, Maya, Rafi, Coco, Dottie, Kutty, Manny, Rosie, Rusty, Pogo, Battery, Laali, Flora, Wally, Karki, Karia, Phil, Cheeto, Sunny, Softy, Tiger, Pompom, Jack, Nymeria, Sherry, Manju - your unconditional and boundless love means the world to me. To Didi and Vineet, thank you for always being there for me, making sure I have a home away from home, and reminding me time and again that this, too, shall pass. I sincerely thank Anand's mom and dad for their constant support, love, and motivation over the last few years. I am truly grateful to have you in my life. Words will not be enough to thank my partner, Anand, the cornerstone of my life. Your steadfast belief in me, unconditional support through the ups and downs, and a constant reminder that there is a life outside of Ph.D. have kept me going through the toughest of times. Thank you for always having my back. To Chini, Chani, Popu, and Leo, thank you for loving me boundlessly, irrespective of my publication count. Most importantly, I would like to thank my parents for being my guiding lights and a source of never-ending inspiration. To Mamma, your endless love, dedication to my well-being above everything else, boundless care, and constant prayers mean the world to me. To Pappa, thank you for being my rock, believing in me, and teaching me dedication, hard work, and resilience through example. Your life is truly an inspiration. I dedicate this thesis wholeheartedly to you.

Abstract

Secure multiparty computation (MPC) allows n distrustful parties to jointly compute a function on their inputs while keeping their inputs private. Security should be preserved even in the presence of an external entity referred to as the adversary that controls some parties and coordinates their behavior. Various settings of secure computation have been considered in the literature depending on factors such as the adversarial power and the allowed probability of error in computation. The feasibility and communication complexity of secure computation has been extensively studied for each setting and constitutes a vast literature in the area.

The work in this thesis aims at advancing the research in secure computation for the most demanding setting of perfect security. Perfect security means that the adversary is all-powerful, that is, the protocol cannot rely on any computational assumptions, and that the protocol has zero probability of error. Such protocols come with desirable properties of unconditional, quantum and everlasting security. They guarantee adaptive security (with some caveats [43, 20]) and remain secure under universal composition [84]. The contributions of this thesis toward advancing the landscape of perfectly-secure multiparty computation in various network settings are abstracted out below.

- Our first work targets the primitive of broadcast, which is essential for secure computation. We focus on optimal resilience (i.e., when the number of corrupted parties t is less than a third of the computing parties n), with no setup or cryptographic assumptions in the synchronous network setting. While broadcast with worst case t rounds is impossible, it has been shown [Feldman and Micali STOC'1988, Katz and Koo CRYPTO'2006] how to construct protocols with expected constant number of rounds in the private channel model. However, those constructions have large communication complexity, specifically $\mathcal{O}(n^2L + n^6 \log n)$ expected number of bits transmitted for broadcasting a message of length L . This leads to a significant communication blowup in secure computation protocols in this setting. We substantially improve the communication com-

plexity of broadcast in constant expected time. We also consider parallel broadcast, where n parties wish to broadcast L bit messages in parallel. Our protocol has no asymptotic overhead for $L = \Omega(n^2 \log n)$, which is a common communication pattern in perfectly secure MPC protocols.

- Subsequently, we study secure multiparty computation in the synchronous network setting with perfect security and optimal resilience (less than one-third of the participants are corrupt). The seminal protocols of Ben-Or, Goldwasser, and Wigderson (STOC'1988), and Chaum, Crépeau and Damgård (STOC'1988) laid the foundations of this setting. Since then, there are, in general, two families of protocols: (a) *Efficient but slow*: These protocols have $\mathcal{O}(n \log n)$ communication complexity per multiplication gate. Still, the running time of these protocols is at least $\Theta(n)$ rounds, even if the depth of the circuit is much smaller $D \ll n$, and (b) *Fast but not efficient*: This line of protocols run at $\mathcal{O}(D)$ expected number of rounds, but require (at least) $\Omega(n^4 \log n)$ communication complexity per multiplication gate. We show that it is possible to simultaneously achieve the best of both the families. We prove that perfectly-secure optimally-resilient secure Multi-Party Computation (MPC) for a circuit with C gates and depth D can be obtained in $\mathcal{O}((Cn + n^4 + Dn^2) \log n)$ communication complexity and $\mathcal{O}(D)$ expected time. For $D \ll n$ and $C \geq n^3$, this is the **first** perfectly-secure optimal-resilient MPC protocol with **linear** communication complexity per gate and **constant** expected time complexity per layer.
- Linear communication overhead forms a natural barrier for general secret-sharing-based MPC protocols. Having matched it in the synchronous network setting, we focus on studying secure multiparty computation in the asynchronous setting with perfect security and optimal resilience (less than one-fourth of the participants are malicious). It has been shown that every function can be computed in this model [Ben-OR, Canetti, and Goldreich STOC'1993]. Despite 30 years of research, all protocols in the asynchronous setting require $\Omega(Cn^2 \log n)$ communication complexity for computing a circuit with C multiplication gates. In contrast, for nearly 15 years, in the synchronous setting, it has been known how to achieve $\mathcal{O}(Cn \log n)$ communication complexity [Beerliova and Hirt TCC'2008]. The techniques for achieving this result in the synchronous setting are not known to be sufficient for obtaining an analogous result in the asynchronous setting. We close this gap between synchronous and asynchronous secure computation and show the first asynchronous protocol with $\mathcal{O}(Cn \log n)$ communication complexity for a circuit with C multiplication gates.

- In a concluding work, deviating from the monolithic view of the network, we study the feasibility of network-agnostic secure multiparty computation with perfect security. Traditionally MPC is studied assuming the underlying network is either synchronous or asynchronous. In a network-agnostic setting, the parties are unaware of the underlying network type. It may be either synchronous or asynchronous, without the knowledge of the parties. The feasibility of perfectly-secure MPC in synchronous and asynchronous networks has been settled a long ago. The landmark work of [Ben-Or, Goldwasser, and Wigderson, STOC'88] shows that $n > 3t_s$ is necessary and sufficient for any MPC protocol with n -parties over synchronous network tolerating t_s active corruptions. In yet another foundational work, [Ben-Or, Canetti, and Goldreich, STOC'93] show that the bound for asynchronous network is $n > 4t_a$, where t_a denotes the number of active corruptions. The previous work on network-agnostic protocols [Appan, Chandramouli, and Choudhury, PODC'22] only shows sufficiency for a bound of $n > 3t_s + t_a$. However, the question of its tightness remains unresolved for network-agnostic setting till date. In this work, we resolve this long-standing question. We show that perfectly-secure network-agnostic n -party MPC tolerating t_s active corruptions when the network is synchronous and t_a active corruptions when the network is asynchronous is possible if and only if $n > 2 \max(t_s, t_a) + \max(2t_a, t_s)$.

Publications based on this Thesis

Accepted Papers

[C1] Ittai Abraham, Gilad Asharov, **Shravani Patil**, and Arpita Patra. “Asymptotically free broadcast in constant expected time via packed VSS.” In Theory of Cryptography Conference, pp. 384-414. Cham: Springer Nature Switzerland, 2022.

[C2] Ittai Abraham, Gilad Asharov, **Shravani Patil**, and Arpita Patra. “Detect, Pack and Batch: Perfectly-Secure MPC with Linear Communication and Constant Expected Time.” In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 251-281. Cham: Springer Nature Switzerland, 2023.

[C3] Ittai Abraham, Gilad Asharov, **Shravani Patil**, and Arpita Patra. “Perfect Asynchronous MPC with Linear Communication Overhead.” In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 280-309. Cham: Springer Nature Switzerland, 2024.

Papers in Submission

[C4] **Shravani Patil**, and Arpita Patra. “Perfectly-secure Network-agnostic MPC with Optimal Resiliency.”

Publications outside this Thesis

Accepted Papers

[C1] Aditya Hegde, Nishat Koti, Varsha Bhat Kukkala, **Shravani Patil**, Arpita Patra, and Protik Paul. “Attaining GOD Beyond Honest Majority with Friends and Foes.” In International Conference on the Theory and Application of Cryptology and Information Security, pp. 556-587. Cham: Springer Nature Switzerland, 2022.

[J1] Nishat Koti, **Shravani Patil**, Arpita Patra, and Ajith Suresh. “MPClan: Protocol suite for privacy-conscious computations.” Journal of Cryptology 36, no. 3 (2023): 22.

Papers in Submission

[C2] Anirudh Chandramouli, **Shravani Patil**, Arpita Patra, and Protik Paul. “YOSOfier: A Compiler for YOSO MPC.”

Contents

- Acknowledgements i
- Abstract iv
- Publications based on this Thesis vii
- Publications outside this Thesis viii
- Contents ix
- List of Figures xiv
- List of Tables xv
- 1 Introduction 1**
 - 1.1 Dimensions of MPC 1
 - 1.1.1 Type of Network 1
 - 1.1.2 Computational Power of the Adversary 2
 - 1.1.3 Type of Adversarial Behavior 3
 - 1.1.4 Capacity of Corruption 3
 - 1.1.5 Adaptivity of Corruption 3
 - 1.2 Attributes of MPC 4
 - 1.2.1 Quality 4
 - 1.2.2 Degree of Robustness 4
 - 1.2.3 Resilience 5
 - 1.2.4 Complexity 5
 - 1.3 Summary of the Contributions of this Thesis 5
 - 1.3.1 Perfectly-secure Broadcast in the Synchronous Network Model 6

1.3.2	Perfectly-secure MPC in the Synchronous Network Model	7
1.3.3	Perfectly-secure MPC in the Asynchronous Network Model	9
1.3.4	Perfectly-secure MPC in the Network-agnostic Model	10
2	Preliminaries	12
2.1	Network Model and Security Definition	12
2.1.1	Synchronous Network	12
2.1.2	Asynchronous Network	14
2.2	Notations	16
2.3	Threshold Secret Sharing Scheme	17
2.4	Reed-Solomon Codes	17
2.4.1	Simultaneous Error Detection and Correction in Reed-Solomon Codes	18
2.5	Bivariate Polynomials	19
2.6	Trivariate Polynomials	19
3	Asymptotically Free Broadcast in Constant Expected Time via Packed VSS	20
3.1	Introduction	20
3.1.1	Our Results	22
3.1.2	Applications and Discussions	25
3.1.3	Related Work	27
3.2	Technical Overview	28
3.2.1	Improved Broadcast in Constant Expected Rounds	28
3.2.2	Packed Verifiable Secret Sharing	32
3.2.3	Optimal Gradecast	35
3.3	Preliminaries	36
3.3.1	Network Model and Notations	36
3.3.2	Bivariate Polynomials	37
3.3.3	Finding (n, t) -STAR	37
3.4	Packed Verifiable Secret Sharing	39
3.5	Balanced Gradecast	45
3.5.1	The Gradecast Protocol	46
3.5.2	Making the Protocol Balanced	53
3.5.3	Conclusions	54
3.6	Multi-Moderated Packed Secret Sharing	54
3.6.1	Reconstruction	61

3.7	Oblivious Leader Election	63
3.8	Broadcast	67
3.8.1	Byzantine Agreement	67
3.8.2	Broadcast and Parallel-broadcast	71
4	Detect, Pack and Batch: Perfectly-Secure MPC with Linear Communication and Constant Expected Time	74
4.1	Introduction	74
4.1.1	Related Work	79
4.2	Technical Overview	80
4.2.1	Detectable and Verifiable Secret Sharing	80
4.2.2	Our MPC Protocol	87
4.2.3	Multiplication Triplets with a Dealer	88
4.3	Preliminaries	90
4.3.1	Network Model and Notations	90
4.3.2	Bivariate Polynomials and Secret Embedding	90
4.3.3	Simultaneous Error Correction and Detection of Reed-Solomon Codes	90
4.3.4	Parallel Broadcast	92
4.4	Packed Secret Sharing	92
4.4.1	Sharing Attempt	93
4.4.2	Reconstruction of g -Polynomials in CONFLICTS	99
4.4.3	Reconstruction of f -Polynomials in CONFLICTS	105
4.4.4	Putting Everything Together: Packed Secret Sharing	111
4.5	Batched and Packed Secret Sharing	119
4.5.1	Sharing	123
4.5.2	Reconstruction	124
4.6	Packed Verifiable Triple Sharing	126
4.6.1	The High-Level Idea	127
4.6.2	Reconstructions	130
4.6.2.1	Weak Private Reconstruction	130
4.6.2.2	Public Reconstruction	136
4.6.3	Putting Everything Together: Packed VTS	140
4.7	Batched Verifiable Triple Sharing	148
4.8	Linear Perfectly Secure MPC	159
4.8.1	Secret Reconstruction	159

4.8.2	From the PSS and VTS to MPC	160
4.8.3	The MPC Protocol	163
5	Perfect Asynchronous MPC with Linear Communication Overhead	167
5.1	Introduction	167
5.1.1	Related work	169
5.2	Technical Overview	169
5.2.1	Basic Asynchronous Verifiable Secret Sharing	170
5.2.2	Our Asynchronous Weak-Binding Secret Sharing	173
5.2.3	Our MPC Protocol	178
5.2.4	Multiplication Triplets with a Dealer	180
5.3	Preliminaries	183
5.3.1	Network Model and Notations	183
5.3.2	Asynchronous Broadcast and Agreement on a Core Set	184
5.3.3	Finding a STAR in a Graph	185
5.3.4	Bivariate Polynomials	186
5.3.5	Trivariate Polynomials	186
5.4	Verifiable Packed Bivariate Secret Sharing	187
5.5	Verifying Product Relation	195
5.5.1	Trivariate Polynomial Verification – Functionality	195
5.5.2	Verifying Product Relation using Trivariate Polynomial	198
5.5.3	Trivariate Polynomial Verification – Protocol	202
5.6	Rate-1 Asynchronous Weak-Binding Secret Sharing	211
5.7	Verifiable Triple Sharing	214
5.7.1	Batching for Linear overhead per triple	217
5.8	Linear Perfectly Secure AMPC	218
5.8.1	Secret Reconstruction	218
5.8.2	The complete MPC protocol	220
5.8.2.1	Preparing the Beaver Triples and Input Sharing	220
5.8.2.2	Batched Beaver Multiplication	222
5.8.3	The MPC Protocol	223
6	Perfectly-secure Network-agnostic MPC with Optimal Resiliency	228
6.1	Introduction	228
6.1.1	Related Work	231

CONTENTS

6.2	Technical Overview	231
6.2.1	Weak and Verifiable Secret Sharing	231
6.2.2	Verifiable Triple Sharing	239
6.2.3	Putting it all together: The MPC Protocol	243
6.3	Preliminaries	244
6.3.1	Network Model and Definitions	244
6.3.2	Symmetric Bivariate Polynomials	245
6.3.3	Finding a (n, t) -Star	245
6.3.4	Almost-surely Terminating	245
6.3.5	Simultaneous Error Correction and Detection of Reed-Solomon Codes	246
6.4	Existing Primitives	247
6.4.1	Finding a (n, t_a) -Star	247
6.4.2	Asynchronous Reliable Broadcast (Acast)	248
6.4.3	Byzantine Broadcast (BC)	249
6.4.4	Byzantine Agreement (BA)	250
6.4.5	Agreement on a Common Set (ACS)	251
6.5	Lower Bound	253
6.6	Weak Secret Sharing	255
6.7	Verifiable Secret Sharing	268
6.8	Verifiable Triple Sharing	279
6.9	Preprocessing Phase	285
6.9.1	Private Reconstruction Protocol	285
6.9.2	Beaver's Multiplication Protocol	286
6.9.3	Triple Extraction Protocol	288
6.10	The Complete MPC Protocol	290
7	Conclusion and Open Problems	295
7.1	Open Problems	297
	Bibliography	299

List of Figures

3.1	Roadmap of our building blocks. All lines are compositions, except for the line from Multi-moderated VSS to Packed VSS, which is a white-box modification.	28
4.1	Pictorial depiction of Packed Verifiable Triple Sharing	129

List of Tables

3.1	Comparison of communication complexity of our work with the state-of-the-art broadcast. $1 \times \mathcal{BC}(L)$ refers to the task of a single dealer broadcasting a L -element message. $n \times \mathcal{BC}(L)$ refers to the task of n dealers broadcasting a L -element message in parallel.	23
6.1	Simultaneous error correction and detection	237

Chapter 1

Introduction

Secure multiparty computation (MPC) allows n distrustful parties to jointly compute a function on their private inputs while ensuring the privacy of these inputs. The distrust among parties is modeled as an adversary that can control and coordinate the behavior of up to t parties. These parties controlled by the adversary are referred to as *corrupt*, whereas those outside the adversary's control are considered to be *honest*. Various settings encountered in real-world scenarios are captured via fundamental dimensions of MPC such as the computational power of the adversary, the type of underlying communication network, and the type of corruption, among others. Further, attributes of MPC protocols such as communication and round complexity enable analysis and comparison of protocols within a particular setting. We discuss the dimensions and attributes of MPC before proceeding to state our contributions.

1.1 Dimensions of MPC

MPC has been studied in various settings which are governed by its dimensions. Looking ahead, the work in this thesis primarily focuses on advancing the research on secure computation protocols in various types of communication networks. We overview some of the relevant dimensions below.

1.1.1 Type of Network

Any MPC protocol requires parties to exchange messages with each other. Typically, the underlying communication network is assumed to be a complete network wherein every pair of distinct parties is connected via a point-to-point private and authentic channel. The allowed uncertainty in communication over the network is captured via the level of synchronization. Depending on this, MPC protocols can be classified into four categories as follows:

1. **Synchronous:** In this setting, it is assumed that parties are synchronized via a common global clock, and there is a known finite time bound Δ on the delay of message delivery over any point-to-point channel. This assumption allows an MPC protocol to proceed in synchronized rounds of communication. Additionally, it also provides the guarantee that the messages between any pair of honest parties is delivered within Δ time. Consequently, if some party fails to communicate in a round where it is expected to, it is guaranteed to be corrupt. However, such strong assumptions on message delivery do not adapt well to real-world networks such as the Internet.
2. **Asynchronous:** In the asynchronous network model, there is no assumption of a common global clock. The communication channels can have arbitrary but finite delays, and moreover, the messages may be delivered in an arbitrary order. The only restriction imposed here is that the messages must be delivered eventually. To capture the worst-case scenario, the adversary is allowed to schedule the delivery of messages among parties.
3. **Hybrid or Partially synchronous:** This setting forms the middle-ground between synchronous and asynchronous networks. Here the assumption is that the protocols can proceed in a few synchronous rounds initially, followed by the network being completely asynchronous. An alternative modeling of such a network also considers that the network is asynchronous initially and switches to synchronous after an unknown but finite time referred to as the global stabilization time.
4. **Network-agnostic:** Recently a line of work [34, 36, 13] focuses on protocols which provide best-of-both worlds guarantees. These protocols are designed to be agnostic of the network type and provide the best guarantees possible depending on whether the actual underlying network condition is synchronous or asynchronous. That is if a network agnostic protocol is instantiated in a synchronous network setting, then it provides the guarantees of a synchronous MPC protocol. On the other hand, it ensures security guarantees of an asynchronous MPC protocol when executed over an asynchronous network.

1.1.2 Computational Power of the Adversary

Based on the allowed computational power, the adversary can be categorized as either computationally bounded or unbounded. In the former case, the adversary is considered to be bounded by probabilistic polynomial time computation. Protocols secure against a bounded adversary often rely on computational hardness assumptions and are termed as computational or cryptographically secure protocols. Whereas in the latter case, the adversary is

allowed access to unbounded compute power. Protocols secure against such an adversary are categorized as information-theoretic or unconditionally secure.

1.1.3 Type of Adversarial Behavior

Depending on the type of misbehaviour allowed during the execution of the protocol, an adversary can be classified into the following categories:

1. **Passive or Semi-honest:** This is the most benign variant of adversarial behavior, wherein the corrupt parties are honest-but-curious. That is, the corrupt parties follow the protocol specification honestly, however the adversary tries to learn more information by observing the internal state (input, output, randomness and the messages exchanged during the protocol) of the corrupt parties.
2. **Active or malicious:** This is a stronger type of adversarial behavior wherein the adversary can control and coordinate the behavior of corrupt parties such that they can arbitrarily deviate from the protocol specification in order to learn more information (beyond what is allowed as per the security guarantees). Such type of corruption is also referred to as Byzantine corruption.
3. **Mixed:** A mixed adversary is allowed to simultaneously perform semi-honest and malicious corruptions. The malicious parties can arbitrarily deviate from the protocol specification, and additionally the adversary has access to the internal states of all (semi-honest and malicious) parties.

1.1.4 Capacity of Corruption

Here, we have two categories of adversaries: threshold and non-threshold. In the former case, the number of parties that can be corrupted by an adversary is bounded by a publicly known threshold t . An MPC protocol in this setting is secure as long as at most t parties are corrupted by the adversary. On the other hand, the corruption ability of a non-threshold adversary is determined by a publicly known adversary structure. An adversary structure is a collection of subsets of parties such that an adversary is allowed to corrupt any one of the subsets from the collection.

1.1.5 Adaptivity of Corruption

An adversary can be broadly categorized into two types depending on its ability to dynamically corrupt parties during the execution of the protocol based on the information it learns.

1. Static: A static adversary decides on the set of parties to be corrupted before the execution of the protocol.
2. Adaptive: In contrast to the static adversary, an adaptive adversary decides on which party to corrupt during the execution of the protocol. This allows the adversary to choose the parties to be corrupted based on the information it learns over time during the protocol execution.

1.2 Attributes of MPC

Once the setting of MPC is determined by fixing the dimensions described above, protocols within this setting can be compared and analyzed based on various attributes that assess their quality and efficiency. The attributes relevant to the work in this thesis are described below.

1.2.1 Quality

As described earlier, MPC protocols can be segregated broadly into two categories, computational and information-theoretic depending on the computational power of the adversary. The former category of computational protocols considers a computationally bounded adversary and additionally allows a non-zero but negligible probability of error in the security guarantees of the protocol. Whereas information-theoretic protocols can be further classified into two types: *statistically-secure* and *perfectly-secure*. Statistically secure protocols tolerate a computationally unbounded adversary; however, such protocols may have a non-zero but negligible probability of error. In contrast, perfect security means that the adversary is computationally unbounded and the protocol has zero probability of error.

1.2.2 Degree of Robustness

Based on the degree of robustness, MPC protocols can be classified into the following three categories:

1. Abort: This is the weakest notion of security, where an adversary can prevent the honest parties from obtaining the output by terminating the protocol upon receiving it.
2. Fair: In this case, the adversary receives the protocol output if and only if the honest parties do, thus ensuring fairness.
3. Guaranteed Output Delivery (GOD): This is the strongest security notion that guarantees that the honest parties receive the output irrespective of the adversary's strategy.

1.2.3 Resilience

Resilience refers to the maximum number of corrupt parties that can be tolerated by a protocol while ensuring security. It is typically captured by the allowed threshold of corruption t . The most common categorization of protocols based on their resilience is that of honest majority ($t < n/2$) and dishonest majority ($t < n$). Several feasibility and impossibility results for resilience are known in the literature conditioned on the setting of MPC and the desired quality. The results relevant to the work in this thesis are provided in the subsequent discussion.

1.2.4 Complexity

The optimality of MPC protocols is measured based on the following three crucial parameters:

1. Round complexity: It measures the number of rounds required by an MPC protocol. This is relevant to the synchronous protocols, which proceed in a sequence of rounds synchronized by the common global clock.
2. Communication complexity: This measures the total number of bits of communication performed by the honest parties during protocol execution.
3. Computational complexity: This captures the amount of computation performed by honest parties in the protocol. Concretely, it can be measured by the number and type of fundamental mathematical operations, the runtime and the throughput of a protocol.

Throughout this thesis, we consider a computationally unbounded, malicious, static, and threshold adversary. In particular, this thesis focuses on perfectly-secure protocols with guaranteed output delivery in various network settings.

1.3 Summary of the Contributions of this Thesis

The objective of this thesis is to enhance the existing knowledge by making significant contributions to the study of perfectly-secure protocols within synchronous, asynchronous, and network-agnostic settings. Our emphasis lies on perfect security owing to its robust assurances. Additionally, it is noteworthy that protocols with perfect security remain adaptively secure (with some caveats [43, 20]) and secure under universal composition [84]. This immediately implies that while this thesis encompasses analysis under a static adversary, the adaptive security of these protocols can be inferred from the aforementioned results. The questions addressed in this thesis are primarily theoretical in nature, nevertheless, they have

a direct impact on the scalability and efficiency of protocols in practice. A vast body of literature exists for the well-studied settings of synchronous and asynchronous protocols. We advance the research in this area by resolving some of the long-standing open questions focused on the communication complexity of perfectly-secure protocols. In addition to being theoretically interesting, our results ensure efficient scalability of MPC protocols. In contrast, the interest in the network-agnostic setting has been fairly recent. Here, our work establishes new lower bounds on resilience and also provides matching upper bounds.

1.3.1 Perfectly-secure Broadcast in the Synchronous Network Model

A common practice in designing secure protocols is to describe the protocol in the broadcast-hybrid model, i.e., to assume the availability of a *broadcast channel*. Such a channel allows a distinguished party to send a message while guaranteeing that all parties receive and agree on the same message. Assuming the availability of a broadcast channel is reasonable only in a restricted setting, for instance, when the parties are geographically close and can use radio waves. In most settings, particularly when executing the protocol over the Internet, parties have to implement this broadcast channel over point-to-point channels.

The cost associated with the implementation of the broadcast channel is often neglected when designing secure protocols. In some settings, the implementation overhead is a real obstacle in practice. In our first work, we focus on designing a broadcast protocol for the most demanding setting: **perfect security with optimal resilience**. Optimal resilience means that the number of parties that the adversary controls is bounded by $t < n/3$, where n is the total number of parties. This bound is known to be tight, as a perfectly-secure broadcast protocol tolerating $n/3$ corrupted parties or more is impossible to construct [85, 94], even when a constant error probability is allowed [2].

There are, in general, two approaches for implementing broadcast in our setting. These approaches provide an intriguing tradeoff between communication and round complexity:

1. **Efficient but slow:** For broadcasting a single bit, the first approach [53, 33] requires $\mathcal{O}(n^2)$ bits of communication complexity, which is asymptotically optimal for any deterministic broadcast protocols [64], or in general, $\mathcal{O}(nL + n^2 \log n)$ bits for broadcasting a message of size L bits via a perfect broadcast extension protocol [47].¹ This comes at the expense of having $\Theta(n)$ rounds.
2. **Fast but not efficient:** The second approach, originated by the seminal work of Feld-

¹Broadcast extension protocols handle long messages efficiently at the cost of a small number of single-bit broadcasts.

man and Micali [66], followed by substantial improvements and simplifications by Katz and Koo [82], requires significant communication complexity of $\mathcal{O}(n^6 \log n)$ bits in expectation for broadcasting just a single bit, or $\mathcal{O}(n^2 L + n^6 \log n)$ bits for a message of L bits. However, they work in *expected constant number of rounds*. Using broadcast extension of [89] we can bring the asymptotic cost to $\mathcal{O}(nL) + E(\mathcal{O}(n^7 \log n))$ bits. However, the minimum message size to achieve this $L = \Omega(n^6 \log n)$. This is prohibitively high even for $n = 100$.

An interesting question exploring the tradeoff between communication and round complexity is thus as follows:

Is there a perfectly secure, optimally-resilient synchronous broadcast with $\mathcal{O}(nL) + E(\mathcal{O}(\text{poly}(n)))$ communication complexity and expected $\mathcal{O}(1)$ rounds for broadcasting L bits such that $\text{poly}(n) \ll n^7 \log n$?

Or is the tradeoff between communication and round complexity unavoidable?

Our Contributions. We provide a significant improvement in the communication complexity of broadcast with perfect security and optimal resilience in the presence of a *static adversary*. Specifically, the expected communication complexity of our protocol is $\mathcal{O}(nL + n^4 \log n)$. For messages of length $L = \Omega(n^3 \log n)$, our broadcast has no asymptotic overhead (up to expectation), as each party has to send or receive $\mathcal{O}(n^3 \log n)$ bits. We also consider parallel broadcast, where n parties wish to broadcast L bit messages in parallel. Our protocol has no asymptotic overhead for $L = \Omega(n^2 \log n)$, which is a common communication pattern in perfectly-secure MPC protocols. For instance, it is common that all parties share their inputs simultaneously at the same round, and verifiable secret sharing protocols require the dealer to broadcast a total of $\mathcal{O}(n^2 \log n)$ bits. As an independent interest, our broadcast is achieved by a *packed verifiable secret sharing*, a new notion that we introduce. We show a protocol that verifies $\mathcal{O}(n)$ secrets simultaneously with the same cost of verifying just a single secret. This improves by a factor of n the state-of-the-art verifiable secret sharing, a pivotal building block in secure computation.

1.3.2 Perfectly-secure MPC in the Synchronous Network Model

Having designed the building block of broadcast, our subsequent work focuses on perfectly-secure MPC protocols in the synchronous network setting where the bound $t < n/3$ on optimal resilience carries forward [85, 94, 29]. The seminal protocols of Ben-Or, Goldwasser, and Wigderson [29], and Chaum, Crépeau and Damgård [46] led the foundations of this setting. Since then, there are, in general, two families of protocols:

1. **Efficient but slow:** These protocols [80, 26, 76] ([26] test-of-time award) have $\mathcal{O}(n \log n)$ communication complexity per multiplication gate. Still, the running time of these protocols is at least $\Theta(n)$ rounds, even if the depth of the circuit is much smaller $D \ll n$. The protocol of [76] requires $\mathcal{O}(n^3 \log n + Cn \log n)$ bits of point-to-point communication and n sequential invocations of broadcast of $\mathcal{O}(\log n)$ bits each, with $\Omega(n + D)$ rounds. Using the broadcast implementation of [7], this translates to $\mathcal{O}(n^5 \log n + Cn \log n)$ bits communication complexity and $\Omega(n + D)$ expected number of rounds for a circuit with C multiplication gates and depth D . Alternatively, using the implementation of [33, 53], the protocol can be more efficient, but even more slower: $\mathcal{O}(n^3 \log n + Cn \log n)$ bits communication complexity and $\Omega(n^2 + D)$ number of rounds.
2. **Fast but not efficient:** This line of protocols [29, 46, 73, 56, 19, 6] run at $\mathcal{O}(D)$ expected number of rounds, but require $\Omega(n^4 \log n)$ communication complexity per multiplication gate. In the broadcast hybrid model, the state-of-the-art protocol of [6] requires $\mathcal{O}(n^3 \log n)$ bits of communication complexity over point-to-point channels and $\mathcal{O}(n^3 \log n)$ bits broadcast, in $\mathcal{O}(D)$ number of rounds. That is, it requires $\Omega(Cn^4 \log n)$ communication complexity and $\mathcal{O}(D)$ expected number of rounds using the broadcast implementation of [7]. Using [33, 53] for implementing the broadcast, the number of rounds is increased to $\Omega(n + D)$.

Given the above two classes of protocols, an interesting problem has remained open for a long time:

*Is there a perfectly secure, optimally-resilient synchronous MPC with $\mathcal{O}(n \log n)$ communication complexity per multiplication gate and expected $\mathcal{O}(D)$ rounds?
Or is the tradeoff between communication and round complexity unavoidable?*

Our Contributions. Our main result is that it is possible to simultaneously achieve the best of both families. For the first time, we provide a perfectly-secure, optimally-resilient MPC protocol that has **both** $\mathcal{O}(n \log n)$ communication complexity per multiplication gate and $\mathcal{O}(D)$ expected round complexity. For $D \ll n$ and $C \geq n^3$, this is the **first** perfectly-secure optimal-resilient MPC protocol with **linear** communication complexity per gate and **constant** expected round complexity per layer in the circuit.

One salient part of our technical contribution is centered around a new primitive we call *detectable secret sharing*. It is perfectly-hiding, weakly-binding, and has the property that either reconstruction succeeds, or $\mathcal{O}(n)$ parties are (privately) detected. On the one hand, we show that detectable secret sharing is sufficiently powerful to generate multiplication triplets

needed for MPC. On the other hand, we show how to share p secrets via detectable secret sharing with communication complexity of just $\mathcal{O}(n^4 \log n + p \log n)$. When sharing $p \geq n^4$ secrets, the communication cost is amortized to just $\mathcal{O}(1)$ field elements per secret.

Our second technical contribution is a new verifiable secret sharing protocol that can share p secrets at just $\mathcal{O}(n^4 \log n + pn \log n)$ communication complexity. When sharing $p \geq n^3$ secrets, the communication cost is amortized to just $\mathcal{O}(n)$ field elements per secret. This further improves our protocol from the previous work by a factor of n .

1.3.3 Perfectly-secure MPC in the Asynchronous Network Model

In the synchronous model of MPC, the assumption is that all messages sent between honest parties arrive after some known bounded delay. This delay bound needs to be fixed in advance and must hold for the lifetime of the system. Fixing a large delay bound may cause the protocol to be inefficient and slow. More worrisome, using a delay that is smaller than the actual delay the adversary can impose may lead to non-termination. In many real world settings it is very hard to guess in advance a bound on the maximum delay the adversary can impose.

These issues in the above setting are mitigated by the category of protocols in the *asynchronous* model, where each message sent between honest parties arrives after some finite delay. This model allows protocols to dynamically adjust to any adversarial network conditions, and obtain termination (with probability 1) even under very powerful adversaries that can adaptively manipulate network delays.

Thus, in this subsequent work, we consider the most demanding setting: *perfect security with optimal resilience in the asynchronous model*. From the lower bound of [30, 31, 8], perfect security implies that the number of corruptions in this setting is at most $t < n/4$, so optimal resilience is when $n = 4t + 1$ (this is in contrast to $n = 3t + 1$ in the synchronous setting). The seminal work of [30, 40] obtains perfect security with optimal resilience in the asynchronous model.

In the perfectly secure, optimally-resilient synchronous model, $\mathcal{O}(n \log n)$ communication complexity per multiplication gate was obtained nearly 15 years ago by the work of [26] which was then improved by our prior work [9] to reduce the round complexity from $\mathcal{O}(D+n)$ to expected $\mathcal{O}(D)$ for circuits of depth D . Linear communication complexity per multiplication gate seems to be a natural barrier. While there is no lower bound, getting $o(n)$ per multiplication gate seems to require fundamentally different techniques and comes at the cost of trading off optimal threshold (e.g., see [59]).

Progress in the (perfectly secure, optimally resilient) asynchronous model over the last 30

years has been slower. The work of [30] obtained $\mathcal{O}(n^6 \log n)$ per gate. [101, 95] improve to $\mathcal{O}(n^5 \log n)$ per gate. [25] improves to $\mathcal{O}(n^3 \log n)$ per gate. The best current bounds are by [92, 93] that obtained $\mathcal{O}(n^2 \log n)$ communication complexity per multiplication gate. A natural question remained open for 30 years:

Is there a perfectly secure, optimally-resilient asynchronous MPC with $\mathcal{O}(n \log n)$ communication complexity per multiplication gate?

Or is there an inherent lower bound due to asynchrony?

Our Contributions. Our main result is a perfectly secure, optimally-resilient asynchronous MPC protocol that achieves $\mathcal{O}(n \log n)$ communication per multiplication gate. We close the gap between synchronous and asynchronous secure computation and show the first asynchronous protocol with $\mathcal{O}(Cn \log n)$ communication complexity for a circuit with C multiplication gates. Our main technical contribution is an asynchronous weak binding secret sharing that achieves rate-1 communication (i.e., $\mathcal{O}(1)$ -overhead per secret). To achieve this goal, we develop new techniques for the asynchronous setting, including the use of *trivariate polynomials* as opposed to the traditional bivariate polynomials used in verifiable secret sharing.

1.3.4 Perfectly-secure MPC in the Network-agnostic Model

So far, the work in this thesis has the traditional, monolithic view of the network. The protocols are designed assuming either a purely synchronous or purely asynchronous network; thus, the parties are aware of the network conditions. Deviating from this traditional approach of modeling the network, a line of research focuses on the scenario where parties are unaware of the network type [34, 36, 63, 13]. The requirements of both synchronous and asynchronous networks must be captured by a single protocol while ensuring security. Protocols designed in this setting are often referred to as *network-agnostic* protocols. While the prior two models had been at the center of study for more than three decades, the latter model has been gaining a lot of traction recently due to its theoretical challenges and practical importance. Having worked on the synchronous and asynchronous network models separately, focusing on network-agnostic protocols with perfect security followed naturally. Hence, we conclude the work in this thesis with the study of network-agnostic MPC with perfect security.

As cited earlier, the feasibility questions for perfectly-secure MPC for synchronous and asynchronous settings have been settled a long ago. While the landmark works of [94, 29] show that perfectly-secure MPC in the synchronous setting tolerating t_s active corruption is possible if and only $t_s < n/3$, it is known that perfect security in the asynchronous setting can

be achieved as long as the number of corrupt parties is $t_a < n/4$ [30, 31, 5]. The feasibility question of perfectly-secure network-agnostic MPC is still alluding. The work of [13] shows sufficiency of such a protocol with $n > 3t_s + t_a$ tolerating t_s active corruptions when the network is synchronous and t_a active corruptions when the network is asynchronous. So far, it is not known if the bound is tight. A natural question thus arises:

Is $n > 3t_s + t_a$ necessary for constructing perfectly-secure network agnostic protocols? Or can we do better?

Our Contributions. Our main result is that we completely settle the feasibility of perfectly-secure network-agnostic MPC. We show that perfectly-secure network-agnostic n -party MPC tolerating t_s active corruptions when the network is synchronous and t_a active corruptions when the network is asynchronous is possible if and only if $n > 2 \max(t_s, t_a) + \max(2t_a, t_s)$. Our main result is obtained via two key components – the necessity and the sufficiency. We prove the lower bound via the technique of proof by contradiction. We identify a function for which the existence of a protocol with $n \leq 2 \max(t_s, t_a) + \max(2t_a, t_s)$ would not allow parties to obtain unanimous output in an asynchronous network, which is otherwise allowed in MPC. Our second contribution thus lies in providing a matching upper bound to prove sufficiency of our lower bound. As mentioned, all the prior work in the perfectly-secure network-agnostic setting considers the non-optimal threshold of $n > 3t_s + t_a$. In our view, the most technically involved contributions here are the weak secret sharing and verifiable triple sharing protocols for the optimal corruption threshold. To design these protocols, we develop new techniques that leverage the fixed time delay if the network is synchronous and rely on a higher number of honest parties when the network is asynchronous while being oblivious to the actual network type.

Chapter 2

Preliminaries

In this chapter, we discuss the requisite background such as the model, definitions, notations and some of the algorithms used throughout this thesis.

2.1 Network Model and Security Definition

The first two chapters in this thesis consider a synchronous network, whereas the third chapter focuses on the asynchronous network. In the former case, we prove the security of our protocols in the standard, standalone simulation-based security model of multiparty computation. We derive universal compositability [42] for free using [84]. Whereas proving security in the latter case is not so straightforward due to the inherent nature of asynchrony and requires additional techniques. Here, we prove the security of our protocols in the simplified universally composable (SUC) setting [45]. Finally, the last chapter considers a network agnostic setting, wherein the network may either be synchronous or asynchronous. This area is fairly recent, and following all the relevant literature, we give property-based proofs for these protocols which are mainly feasibility results.

2.1.1 Synchronous Network

For the first two chapters in this thesis, we consider a synchronous network model where the parties in $\mathcal{P} = \{P_1, \dots, P_n\}$ are connected via pairwise private and authenticated channels. Additionally, for some of our protocols we assume the availability of a broadcast channel, which allows a party to send an identical message to all the parties. The distrust in the network is modelled as a *computationally unbounded* active adversary \mathcal{A} which can maliciously corrupt up to t out of the n parties during the protocol execution and make them behave in an arbitrary manner. We prove security in the stand-alone model for a static adversary. We provide the definitions (which are standard) below. Owing to the results of [44], this

guarantees adaptive security with inefficient simulation. As mentioned, we derive universal composability [42] for free using [84].

We prove the security of our protocols in the standard, standalone simulation-based security model of multiparty computation in the perfect settings [41, 18]. Let $f : (\{0, 1\}^*)^n \rightarrow (\{0, 1\}^*)^n$ be an n -party functionality and let π be an n -party protocol over private and authenticated point-to-point channels and an authenticated broadcast channel. Let \mathcal{A} be the adversary with auxiliary input z , and let $\mathcal{C} \subset \mathcal{P}$ be the set of corrupted parties controlled by it. We define the real and ideal executions:

- **Execution in the real model with protocol π :** In the real model, the parties run the protocol π where the adversary \mathcal{A} controls the parties in \mathcal{C} . The adversary is assumed to be rushing, meaning that in every round it can see the messages sent by the honest parties to the corrupted parties before it determines the message sent by the corrupted parties. The adversary cannot see the messages sent between honest parties on the point-to-point channels. We denote by $\text{Real}_{\mathcal{A}(z), \mathcal{C}}^\pi(\vec{x})$ the random variable consisting of the view of the adversary \mathcal{A} in the execution (consisting of all the initial inputs of the corrupted parties, their randomness and all messages they received), together with the output of all honest parties.
- **Execution in the ideal model:** The ideal model consists of all honest parties, a trusted party and an ideal adversary \mathcal{SJM} , controlling the same set of corrupted parties \mathcal{C} . The honest parties send their inputs to the trusted party. The ideal adversary \mathcal{SJM} receives the auxiliary input z and sees the inputs of the corrupted parties. \mathcal{SJM} can substitute any x_i with any x'_i of its choice (for the corrupted parties) under the condition that $|x'_i| = |x_i|$. Once the trusted party receives (possibly modified) inputs (x'_1, \dots, x'_n) from all parties, it computes $(y_1, \dots, y_n) = f(x'_1, \dots, x'_n)$ and sends y_i to P_i . The output of the ideal execution, denoted as $\text{Ideal}_{\mathcal{SJM}(z), \mathcal{C}}^f(\vec{x})$ is the output of all honest parties and the output of the ideal adversary \mathcal{SJM} .

Definition 2.1.1. *We say that a protocol π is t -secure for a functionality f , if for every adversary \mathcal{A} in the real world, there exists an adversary \mathcal{SJM} in the ideal world such that for every $\mathcal{C} \subset \mathcal{P}$ of cardinality at most t , it must hold that*

$$\{\text{Ideal}_{\mathcal{SJM}(z), \mathcal{C}}^f(\vec{x})\} \equiv \{\text{Real}_{\mathcal{A}(z), \mathcal{C}}^\pi(\vec{x})\}$$

where \vec{x} is chosen from $(\{0, 1\}^*)^n$ such that $|x_1| = \dots = |x_n|$.

Modular composition. We also consider standard f -hybrid model. In the f -hybrid model, the parties can have access to some trusted party that can compute some functionality f for them. See, e.g., [41, 18] for further details.

2.1.2 Asynchronous Network

We consider an asynchronous network where the parties are $\mathcal{P} = \{P_1, \dots, P_n\}$. The parties are connected via pairwise ideal private channels. To model asynchrony, messages sent on a channel can be arbitrarily delayed, however, they are guaranteed to be eventually received after some finite number of activations of the adversary. In general, the order in which messages are received might be different from the order in which they were sent. Yet, to simplify notation and improve readability, we assume that the messages that a party receives from a channel are guaranteed to be delivered in the order they were sent. This can be achieved using standard techniques – counters, and acknowledgements, and so we just make this simplification assumption.

We prove our protocols in this simplified universally composable setting (SUC), which is a simplified UC model aimed for modeling secure protocols, formalized by Canetti, Cohen and Lindell [45], and implies UC security. We briefly review the definitions, but many details are left out, see [45] for additional information.

Main difference from SUC. The SUC model allows the adversary to also drop messages, and the adversary is not limited to eventually deliver all messages. To model “eventual delivery” (which is the essence of the asynchronous model), we limit the capabilities of the adversary and quantify over adversaries that eventually transmit each message in the network (i.e., they do not drop messages). Formally, any message sent must be delivered after some finite number of activations of the adversary.

As in SUC, the parties are modeled as interactive Turing machines, with code tapes, input tapes, outputs tapes, incoming communication tapes, outgoing communication tape, random tape and work tape.

Communication. In each execution there is an *environment* \mathcal{Z} , an *adversary* \mathcal{A} , participating *parties* P_1, \dots, P_n , and possibly an *ideal functionality* \mathcal{F} and a *simulator* \mathcal{S} . The parties, adversary and ideal functionality are connected in a star configuration, where all communication is via an additional *router machine* that takes instructions from the adversary. That is, each entity has one outgoing channel to the router and one incoming channel. When P_i sends a message to P_j , it sends it to the router, and the message is stored by the router. The router delivers to the adversary a general information about the message (i.e., “a header” but

not the “content”. That is, the adversary can know the type of the message and its size, but cannot see its content). When the adversary allows the delivery of the message, the router delivers the message to P_j . As mentioned, we quantify only over all adversaries that eventually deliver all messages. In particular, even in an execution with an ideal functionality, communication between the parties and this functionality is done via the router machine and is subject to (finite) delivery delays imposed by the adversary.

Note that the router machine is also part of the ideal model. When the functionality gives for instance, some output to party P_j , then this is performed via the router, and the simulator is notified. Thus, if the adversary, for instance, delays the delivery of the output of some party P_j , we do not explicitly mention that in the functionality (e.g., “wait to receive OK_j from the adversary and then deliver the output to P_j ”), yet it is captured by the model.

Finally, the environment \mathcal{Z} communicates with the adversary directly and not via the router. In particular, the environment can communicate only with the adversary (and it cannot communicate even with the ideal functionality \mathcal{F}). In addition, \mathcal{Z} can *write* inputs to the honest parties’ input tapes and can *read* their output tapes.

- **Execution in the ideal model:** In the ideal model we consider an execution of the environment \mathcal{Z} , dummy parties P_1, \dots, P_n , the router, a functionality \mathcal{F} and a simulator \mathcal{S} . In the ideal model with a functionality \mathcal{F} the parties follow a fixed ideal-model protocol. The environment is first activated with some input z . The environment delivers the inputs to the dummy honest parties, which forward the inputs to the functionality (recall that this is done via the router, which then gives some leakage about the message header to \mathcal{S} , which can adaptively delay the delivery by any finite amount). Moreover, \mathcal{Z} can also give some initial inputs to the corrupted parties via \mathcal{S} . At a later stage where the dummy parties receive output from the functionality \mathcal{F} , they just write the outputs on their output tapes (and \mathcal{Z} can read those outputs). The simulator \mathcal{S} can send messages to \mathcal{Z} and to the functionality \mathcal{F} . The simulator cannot directly communicate with the participating parties. We stress that in the ideal model, the simulator \mathcal{S} interacts with \mathcal{Z} in an online way, and the environment can essentially read the outputs of the honest parties, and query the simulator (i.e., can see the view of the adversary) at any point of the execution. At the end of the interaction, \mathcal{Z} outputs some bit b . We denote by $\text{IDEAL}_{\mathcal{F}, \mathcal{S}, \mathcal{Z}}(z)$ an execution of this ideal model of the functionality \mathcal{F} with a simulator \mathcal{S} and environment \mathcal{Z} , which starts with an input z .
- **Execution in the real model with protocol π :** In the real model, there is no ideal functionality and the participating parties are \mathcal{Z} , the parties P_1, \dots, P_n , the router and

the real-world adversary \mathcal{A} . The environment is first activated with some input z , and it can give inputs to the honest parties, as well as some initial inputs to the corrupted parties controlled by the adversary \mathcal{A} . The parties run the protocol π as specified, while the corrupted parties are controlled by \mathcal{A} . The environment can see at any point the outputs of the honest parties, and communicate directly with the adversary \mathcal{A} (and see, without loss of generality, its partial view). At the end of the execution, the environment outputs some bit b . We denote by $\text{REAL}_{\pi, \mathcal{A}, \mathcal{Z}}(z)$ an execution of this real model with the protocol π , the real-world adversary \mathcal{A} and the environment \mathcal{Z} , which starts with some input z .

Definition 2.1.2. *We say that an adversary \mathcal{A} is an asynchronous adversary if any message that it receives from the router, it allows its delivery within some finite number of activations of \mathcal{A} .*

Definition 2.1.3. *Let π be a protocol and let \mathcal{F} be an ideal functionality. We say that π securely computes \mathcal{F} in the asynchronous setting if for every real-model asynchronous adversary \mathcal{A} there exists an ideal-world adversary \mathcal{S} that runs in polynomial time in \mathcal{A} 's running time, such that for every \mathcal{Z} :*

$$\{\text{IDEAL}_{\mathcal{F}, \mathcal{S}, \mathcal{Z}}(z)\}_z \equiv \{\text{REAL}_{\pi, \mathcal{A}, \mathcal{Z}}(z)\}_z$$

Hybrid Model and Composition We also consider a hybrid model where the parties follow some protocol π as in the real model but also have access to some ideal functionality \mathcal{F} , and the protocol instructs the parties to send messages to that ideal functionality and how to process its response. As previously, all communication is performed via the router. We denote the output of \mathcal{Z} from a hybrid execution of π with ideal calls to \mathcal{F} as $\text{HYBRID}_{\pi, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}}(z)$, where $\mathcal{A}, \mathcal{Z}, z$ are as above. We call this as \mathcal{F} -hybrid model.

The composition theorem states that if a protocol π realizes \mathcal{F} in the \mathcal{G} -Hybrid model, and a protocol ρ realizes \mathcal{G} in the plain model, then the protocol π^ρ realizes \mathcal{F} in the plain model. To clarify, when the parties in π call \mathcal{G} in the protocol π , in the protocol π^ρ the parties will invoke the code of the protocol ρ . This is a difference between the SUC and UC model, in which in the UC model, each subprotocol is invoked as a separate interactive Turing machine, which introduces some extra complexity. See [45] for elaborated discussion.

2.2 Notations

Our protocols are defined over a finite field \mathbb{F} where $|\mathbb{F}| > n$ holds throughout this thesis. We denote the elements by $\{0, 1, \dots, n\}$. However, the actual size $|\mathbb{F}|$ of the field required varies depending on the setting and the protocols. We discuss these details in the relevant chapters

to keep the preliminaries generic at the outset of this thesis. We use $\langle v \rangle$ to denote the degree- t Shamir-sharing of a value v among parties in \mathcal{P} , and $\langle v \rangle_i$ to denote party P_i 's share.

2.3 Threshold Secret Sharing Scheme

We consider a threshold adversary throughout this thesis; consequently, our protocols rely on a threshold secret sharing scheme. Informally, a k -out-of- n or a (k, n) -threshold secret sharing scheme shares a secret among n parties such that no coalition of up to k parties can learn (any information about) the secret, whereas any group of $k + 1$ or more parties can completely reconstruct the secret. Here k refers to the threshold of the secret sharing scheme. Shamir sharing is a specific instantiation of a threshold secret sharing scheme, which we use throughout this thesis. Unless stated otherwise, for most parts of the thesis we use $k = t$, where t is the maximum number of parties that can be corrupted by the adversary.

Definition 2.3.1. A k -out-of- n or (k, n) -threshold secret sharing scheme defined over a message space \mathcal{M} for a set of parties $\mathcal{P} = \{P_1, \dots, P_n\}$ consists of two algorithms:

1. *Sharing (Share): It is a randomized algorithm which takes as input a message or secret $m \in \mathcal{M}$, and gives a sequence of shares (s_1, \dots, s_n) as the output.*
2. *Reconstruction (Rec): It is a deterministic algorithm that takes as input a set of $k + 1$ or more shares and gives a message or secret value as its output.*

Any threshold secret sharing scheme satisfies the following requirements:

- *Correctness: Any set of $k + 1$ parties can reconstruct the correct secret using Rec. That is, $\forall U \subseteq \mathcal{P}$ such that $|U| \geq k + 1$, $\text{Rec}(\{s_i\}_{P_i \in U}) = m$.*
- *Privacy: Any set of k or fewer parties cannot learn any information about the underlying secret. Specifically, $\forall s, s' \in \mathcal{M}, \forall U \subseteq \mathcal{P}$ such that $|U| \leq k$, it holds that the following distributions are identical $\{\{s_i\}_{P_i \in U}\} \equiv \{\{s'_i\}_{P_i \in U}\}$ where $(s_1, \dots, s_n) \leftarrow \text{Share}(s)$ and $(s'_1, \dots, s'_n) \leftarrow \text{Share}(s')$.*

2.4 Reed-Solomon Codes

Reed-Solomon (RS) codes are a class of non-binary error-correcting codes defined over a finite field characterized by three parameters: alphabet size (q), message length ($k + 1$) and block length n . There are various methods for encoding Reed-Solomon codes, each yielding different representations of the set of all code words. One such perspective, originally

introduced by Reed and Solomon [99], every code word in the Reed-Solomon code comprises a series of function values derived from a polynomial with a degree of up to k . To construct a code word of a message, the symbols of the message, each from an alphabet of size q such as a finite field of size q , are interpreted as the coefficients of a polynomial p with a degree of up to k . This polynomial p is then evaluated at $n \leq q$ distinct points within the finite field \mathbb{F} consisting of q elements. The resulting sequence of values constitutes the corresponding code word. Throughout the thesis, we use this interpretation of RS codes which naturally aligns with our use of Shamir secret sharing. Specifically, we utilize the error detection and correction capabilities of RS codes to ensure correct sharing and reconstruction of secrets in several primitives that we design such as verifiable secret sharing (Chapters 4, 6), detectable secret sharing (Chapter 4) and weak secret sharing (Chapter 6). We give the relevant details regarding error detection and correction below and defer the specific parameters used to the respective chapters.

2.4.1 Simultaneous Error Detection and Correction in Reed-Solomon Codes

Let C be a Reed-Solomon code word of length n , corresponding to a k -degree polynomial (containing $k + 1$ coefficients). Assume that at most t errors can occur in C . Let \bar{C} be the word after introducing error in C in at most t positions. Let the distance between C and \bar{C} be s where $s \leq t$. Then there exists an *efficient* decoding algorithm that takes \bar{C} and a pair of parameters (e, e') as input, such that $e + e' \leq t$ and $n - k - 1 \geq 2e + e'$ hold and gives one of the following as output:

1. Correction: output C if $s \leq e$, i.e. the distance between C and \bar{C} is at most e ;
2. Detection: output “more than e errors” otherwise.

Note that detection does not return the error indices, rather it simply indicates error correction fails due to the presence of more than correctable (i.e. e) errors. The above property of RS codes is traditionally referred to as *simultaneous error correction and detection*. In fact the bounds, $e + e' \leq t$ and $n - k - 1 \geq 2e + e'$, are known to be necessary. Formally:

Theorem 2.4.1 ([49, 87]). *Let C be a Reed-Solomon (RS) code word of length n , corresponding to a k -degree polynomial (containing $k + 1$ coefficients). Let \bar{C} be a word of length n such that the distance between C and \bar{C} is at most t . Then RS decoding can correct up to e errors in \bar{C} to reconstruct C and detect the presence of up to $e + e'$ errors in \bar{C} if and only if $n - k - 1 \geq 2e + e'$ and $e + e' \leq t$.*

2.5 Bivariate Polynomials

A bivariate polynomial of degree p in x and degree q in y , also referred to as (p, q) -bivariate polynomial is of the form:

$$S(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^p \sum_{j=0}^q a_{i,j} \mathbf{x}^i \mathbf{y}^j .$$

Claim 2.5.1 (Interpolation). *Let $H \subset [n]$ be a set of cardinality $q + 1$ and let $(f_h(\mathbf{x}))_{h \in H}$ be $q + 1$ univariate polynomials of degree at most p . Then, there exists a unique bivariate polynomial $S(\mathbf{x}, \mathbf{y})$ of degree p in x and degree q in y satisfying for every $h \in H$: $S(\mathbf{x}, h) = f_h(\mathbf{x})$.*

Following the vast literature on perfectly-secure protocols, we extensively make use of bivariate polynomials to design verifiable secret sharing schemes in different network scenarios (Chapters 3, 4, 5, 6). The exact degree of polynomials we use is customised to the type of underlying network considered and hence we defer these details to the relevant chapters.

2.6 Trivariate Polynomials

A trivariate polynomial of degree p, q, r in variables x, y, z respectively is of the form:

$$S(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{i=0}^p \sum_{j=0}^q \sum_{k=0}^r a_{i,j,k} \mathbf{x}^i \mathbf{y}^j \mathbf{z}^k .$$

Claim 2.6.1 (Interpolation). *Let $H \subset [n]$ be a set of cardinality $r + 1$, and let $(S_h(\mathbf{x}, \mathbf{y}))_{h \in H}$ be $r + 1$ bivariate polynomials of degree at most p in x and at most q in y . Then, there exists a unique trivariate polynomial $S(\mathbf{x}, \mathbf{y}, \mathbf{z})$ of degree p, q, r in x, y, z respectively such that for every $h \in H$:*

$$S(\mathbf{x}, \mathbf{y}, h) = S_h(\mathbf{x}, \mathbf{y})$$

In our work on perfectly secure protocols in the asynchronous network (Chapter 5), we encounter that the bivariate polynomials traditionally used in constructing weak and verifiable secret sharing protocols, do not suffice in achieving our goal of an MPC protocol with linear communication complexity per gate. We circumvent this problem by using trivariate polynomials for sharing, and give the details of the additional properties offered and the exact degree of the polynomial in the corresponding chapter.

We remark that this chapter outlines the notation and preliminary concepts that are consistently used throughout this thesis. Any additional notation or preliminaries specific to each individual work will be introduced in their respective chapters. The subsequent chapters will present a detailed description of the contributions of this thesis.

Chapter 3

Asymptotically Free Broadcast in Constant Expected Time via Packed VSS

In this chapter, we discuss our protocol for perfectly-secure broadcast in the synchronous network. In the process, we introduce the building blocks such as verifiable secret sharing and gradedcast for which we provide improved constructions.

3.1 Introduction

As mentioned earlier, a common practice in designing secure protocols is to describe the protocol in the broadcast-hybrid model, i.e., to assume the availability of a *broadcast channel*. Such a channel allows a distinguished party to send a message while guaranteeing that all parties receive and agree on the same message. In most settings, particularly when executing the protocol over the Internet, parties have to implement this broadcast channel over point-to-point channels. The cost associated with the implementation of the broadcast channel is often neglected when designing secure protocols and indeed it forms the overhead in some settings in practice. In this chapter, we focus on the most demanding setting: **perfect security with optimal resilience** ($t < n/3$) [85, 94].

Asymptotically-free broadcast. What is the best implementation of broadcast that we can hope for? For broadcasting an L bit message, consider the ideal trusted party that implements an “ideal broadcast”. Since each party has to receive L bits, the total communication is $\mathcal{O}(nL)$. To avoid bottlenecks, we would also prefer *balanced* protocols where all parties have to communicate roughly the same number of bits, i.e., $\mathcal{O}(L)$, including the sender.

Regarding the number of rounds, it has been shown that for any broadcast protocol with perfect security there exists an execution that requires $t+1$ rounds [69]. Therefore, a protocol

that runs in strict constant number of rounds is impossible to achieve. The seminal works of Rabin and Ben-Or [96, 27] demonstrated that those limitations can be overcome by using randomization. We define *asymptotically-free broadcast* as a balanced broadcast protocol that runs in *expected* constant number of rounds and with (expected) communication complexity of $\mathcal{O}(nL)$.

There are, in general, two approaches for implementing broadcast in our setting. These approaches provide an intriguing tradeoff between communication and round complexity:

- **Low communication complexity, high number of rounds:** For broadcasting a single bit, the first approach [53, 33] requires $\mathcal{O}(n^2)$ bits of communication complexity, which is asymptotically optimal for any deterministic broadcast protocols [64], or in general, $\mathcal{O}(nL + n^2 \log n)$ bits for broadcasting a message of size L bits via a perfect broadcast extension protocol [47].¹ This comes at the expense of having $\Theta(n)$ rounds.
- **High communication complexity, constant expected number of rounds:** The second approach, originated by the seminal work of Feldman and Micali [66], followed by substantial improvements and simplifications by Katz and Koo [82], requires significant communication complexity of $\mathcal{O}(n^6 \log n)$ bits in expectation for broadcasting just a single bit, or $\mathcal{O}(n^2 L + n^6 \log n)$ bits for a message of L bits.² However, they work in *expected constant number of rounds*.

To get a sense of how the above translates to practice, consider a network with 200ms delay per round-trip (such a delay is relatively high, but not unusual, see [1]), and $n = 300$. Using the first type of protocol, ≈ 300 rounds are translated to a delay of 1 minute. Then, consider for instance computing the celebrated protocol of Ben-Or, Goldwasser and Wigderson [29] on an arithmetic circuit with depth 30. In each layer of the circuit the parties have to use broadcast, and thus the execution would take at least 30 minutes. The second type of protocols require at least $\Omega(n^6 \log n)$ bits of communication. The protocol is balanced and each party sends or receives $n^5 \log n$ bits ≈ 2.4 terabytes. Using 1Gbps channel, this is a delay of 5.4 hours. Clearly, both approaches are not ideal.

This current state of the affairs calls for the design of faster broadcast protocols and in particular, understanding better the tradeoff between round complexity and communication complexity.

¹Broadcast extension protocols handle long messages efficiently at the cost of a small number of single-bit broadcasts.

²Using broadcast extension of [89] we can bring the asymptotic cost to $\mathcal{O}(nL) + E(\mathcal{O}(n^7 \log n))$ bits. However, the minimum message size to achieve this $L = \Omega(n^6 \log n)$. This is prohibitively high even for $n = 100$.

Why perfect security? Our main motivation for studying broadcast is for perfectly secure multiparty computation. Perfect security provides the strongest possible security guarantee which provides unconditional, quantum, and everlasting security. Perfect broadcast is an essential primitive in generic perfectly secure protocols.

Even if we relax our goals and aim for statistical security only, the situation is not much better. Specifically, the best upper bounds that we have are in fact already perfectly secure [53, 33, 82, 90, 89, 47]. That is, current statistically secure results do not help in achieving a better communication complexity vs round complexity tradeoff relative to the current perfect security results. We remark that in the computational setting, in contrast, the situation is much better. Asymptotically-free broadcast with $f < n/2$ can be achieved assuming threshold signatures and setup assumption in constant expected rounds and with $\mathcal{O}(n^2 + nL)$ communication [82, 4, 100].

3.1.1 Our Results

We provide a significant improvement in the communication complexity of broadcast with perfect security and optimal resilience in the presence of a *static adversary*. Towards that end, we also improve a pivotal building block in secure computation, namely, verifiable secret sharing (VSS). Our new VSS has an $\mathcal{O}(n)$ complexity improvement that may be of independent interest. We present our results in a top-down fashion. Our main result is:

Theorem 3.1.1. *There exists a perfectly secure, balanced, broadcast protocol with optimal resilience, which allows a dealer to send L bits at the communication cost of $\mathcal{O}(nL)$ bits, plus $\mathcal{O}(n^4 \log n)$ expected bits. The protocol runs in constant expected number of rounds and assumes private channels.*

Previously, Katz and Koo [82] achieved $\mathcal{O}(n^2L)$ bits plus $\mathcal{O}(n^6 \log n)$ expected number of bits. For messages of size $L = \Omega(n^3 \log n)$ bits, the total communication of our protocol is $\mathcal{O}(nL)$ bits. Thus, we say that our protocol is asymptotically free for messages of size $L = \Omega(n^3 \log n)$ bits. We recall that [82] together with [89] are also asymptotically free albeit only for prohibitively large value of L ($= \Omega(n^6 \log n)$). Table 3.1 compares our work to the state of the art in broadcast protocols.

To get a sense from a practical perspective, for broadcasting a single bit with $n = 300$, our protocol requires each party to send/receive roughly $n^3 \log n \approx 27$ MB (as opposed to ≈ 2.4 terabytes by [82]). Using a 1Gbps channel, this is 200ms. For broadcasting a message of size ≈ 27 MB, each party still has to send/receive roughly the same size of this message, and the broadcast is asymptotically free in that case.

Task	Reference	Total P2P (in bits)	Rounds
$1 \times \mathcal{BC}(L)$	[53, 33]	$\mathcal{O}(n^2L)$	$\mathcal{O}(n)$
	[53, 33] + [47]	$\mathcal{O}(nL + n^2 \log n)$	$\mathcal{O}(n)$
	[82]	$\mathcal{O}(n^2L) + E(\mathcal{O}(n^6 \log n))$	$E(\mathcal{O}(1))$
	[82] + [89]	$\mathcal{O}(nL) + E(\mathcal{O}(n^7 \log n))$	$E(\mathcal{O}(1))$
	Our work	$\mathcal{O}(nL) + E(\mathcal{O}(n^4 \log n))$	$E(\mathcal{O}(1))$
$n \times \mathcal{BC}(L)$	[53, 33]	$\mathcal{O}(n^3L)$	$\mathcal{O}(n)$
	[82]	$\mathcal{O}(n^3L) + E(\mathcal{O}(n^6 \log n))$	$E(\mathcal{O}(1))$
	[82] + [89] ²	$\mathcal{O}(n^2L) + E(\mathcal{O}(n^7 \log n))$	$E(\mathcal{O}(1))$
	Our work	$\mathcal{O}(n^2L) + E(\mathcal{O}(n^4 \log n))$	$E(\mathcal{O}(1))$

Table 3.1: Comparison of communication complexity of our work with the state-of-the-art broadcast.
 $1 \times \mathcal{BC}(L)$ refers to the task of a single dealer broadcasting a L -element message.
 $n \times \mathcal{BC}(L)$ refers to the task of n dealers broadcasting a L -element message in parallel.

Parallel composition of broadcast. In MPC, protocols often instruct the n parties to broadcast messages of the same length L in parallel at the same round. For instance, in the protocol of [29], all parties share their input at the same round, and for verifying the secret, each party needs to broadcast $L = \mathcal{O}(n^2 \log n)$ bits.¹ In fact, the notion of parallel-broadcast goes back to the work of Pease et al. [94]. We have the following extension to our main result:

Corollary 3.1.2. *There exists a perfectly-secure, balanced, parallel-broadcast protocol with optimal resilience, which allows n dealers to send messages of size L bits each, at the communication cost of $\mathcal{O}(n^2L)$ bits, plus $\mathcal{O}(n^4 \log n)$ expected bits. The protocol runs in constant expected number of rounds.*

For message of size $L = \mathcal{O}(n^2 \log n)$ bits, which is common in MPC, our broadcast is asymptotically optimal. We obtain a cost of $\mathcal{O}(n^4 \log n)$ bits in expectation, with expected constant rounds. Note that each party receives $\mathcal{O}(nL)$ bits, and therefore $\mathcal{O}(n^2L) = \mathcal{O}(n^4 \log n)$ bits is the best that one can hope for. Again, the protocol is balanced, which means that each party sends or receives only $\mathcal{O}(nL)$ bits.

For comparison, the other approach for broadcast based on [53, 33, 47] requires total $\mathcal{O}(n^4 \log n)$ bits for this task, but with $\Theta(n)$ rounds. We refer again to Table 3.1 for comparison.

To get a practical sense of those complexities, when $n = 300$ and parties have to broadcast simultaneously messages of size L , our protocol is asymptotically optimal for $L = n^2 \log n \approx 90\text{KB}$.

¹In fact, in each round of the protocol, each party performs $\mathcal{O}(n)$ verifiable secret sharings (VSSs), i.e., it has to broadcast $\mathcal{O}(n^3 \log n)$ bits. In [6] it has been shown how to reduce it to $\mathcal{O}(1)$ VSSs per party, i.e., each party might have to broadcast $\mathcal{O}(n^2 \log n)$.

²Since the broadcast extension protocol of [47] requires $\mathcal{O}(n)$ rounds, combining [82] with [47] results in

Packed verifiable secret sharing. A pivotal building block in our construction, as well as perfectly secure multiparty protocols is *verifiable secret sharing* (VSS), originally introduced by Chor et al. [48]. It allows a dealer to distribute a secret to n parties such that no share reveal any information about the secret, and the parties can verify, already at the sharing phase, that the reconstruction phase would be successful.

To share a secret in the semi-honest setting, the dealer embeds its secret in a degree- t univariate polynomial, and it has to communicate $\mathcal{O}(n)$ field elements. In the malicious setting, the dealer embeds its secret in a bivariate polynomial of degree- t in both variables [29, 67]. The dealer then has to communicate $\mathcal{O}(n^2)$ field elements to share its secret. An intriguing question is whether this gap between the semi-honest (where the dealer has to encode its secret in a structure of size $\mathcal{O}(n)$) and the malicious setting (where the dealer has to encode its secret in a structure of size $\mathcal{O}(n^2)$) is necessary. While we do not answer this question, we show that the dealer can pack $\mathcal{O}(n)$ secrets, simultaneously in one bivariate polynomial. Then, it can share it at the same cost as sharing a single VSS, achieving an overhead of $\mathcal{O}(n)$ per secret. We show:

Theorem 3.1.3. *Given a synchronous network with pairwise private channels and a broadcast channel, there exists a perfectly secure packed VSS protocol with optimal resilience, which has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast for sharing $\mathcal{O}(n)$ secret field elements (i.e., $\mathcal{O}(n \log n)$ bits) in strict $\mathcal{O}(1)$ rounds. The optimistic case (where all the parties behave honestly) does not use the broadcast channel in the protocol.*

The best previous results achieve $\mathcal{O}(n^3 \log n)$ (point-to-point and broadcast) for sharing $\mathcal{O}(n)$ secret elements [29, 67, 83, 16], this is an improvement by a factor of n in communication complexity.

Packing k secrets into one polynomial is a known technique, proposed by Franklin and Yung [71]. It was previously used in Shamir’s secret sharing scheme. However, it comes with the following price: While Shamir’s secret sharing allows protecting against even $n - 1$ corrupted parties, packing k secrets in one polynomial achieves privacy against only $n - k - 1$ parties. In the malicious case, VSS of a single secret is possible only when the number of corruption satisfies $t < n/3$ to begin with, and thus when packing many secrets we do not lose in the resilience of the protocol. The idea of packing many secrets without trading off the allowed threshold of corruption has been explored by Damgård et al. [60]. However, it

linear-round complexity and a worse communication complexity than what the second row ([53, 33] + [47]) provides.

is achieved at the expense of having $\mathcal{O}(n)$ rounds. In contrast, our packed verifiable secret sharing enables packing $\mathcal{O}(n)$ secrets while keeping the threshold exactly the same and ensuring $\mathcal{O}(1)$ round complexity. Compared to a constant round VSS of a single secret, we obtain packed secret sharing completely for free (up to small hidden constants in the \mathcal{O} notation of the above theorem). Lastly, the same result as ours is achieved in the asynchronous setting with the optimal resilience of $t < n/4$ in [50, 51].

Optimal gradecast for $\Omega(n^2)$ messages. Another building block that we improve along the way is gradecast. Gradecast is a relaxation of broadcast introduced by Feldman and Micali [66] (“graded-broadcast”). It allows a distinguished dealer to transmit a message, and each party outputs the message it receives together with a grade $g \in \{0, 1, 2\}$. If the dealer is honest, all honest parties receive the same message and grade 2. If the dealer is corrupted, but some honest party outputs grade 2, it is guaranteed that all honest parties output the same message (though some might have grade 1 only). We show that:

Theorem 3.1.4. *There exists a perfectly secure gradecast protocol with optimal resilience, which allows a party to send a message of size L bits with a communication cost of $\mathcal{O}(nL + n^3 \log n)$ bits and in $\mathcal{O}(1)$ rounds. The protocol is balanced.*

Note that this result is optimal when $L = \Omega(n^2 \log n)$ bits as each party has to receive L bits even in an ideal implementation. Previously, the best gradecast protocol in the perfect security setting [66] required $\mathcal{O}(n^2 L)$ bits of communication.

3.1.2 Applications and Discussions

Applications: Perfect secure computation. We demonstrate the potential speed up of protocols in perfect secure computation using our broadcast. There are, in general, two lines of works in perfectly secure MPC, resulting again in an intriguing tradeoff between round complexity and communication complexity.

The line of work [29, 46, 73, 56, 19, 6] achieves constant round per multiplication and round complexity of $\mathcal{O}(\text{depth}(C))$, where C is the arithmetic circuit that the parties jointly compute. The communication complexity of those protocols results in $\mathcal{O}(n^3 |C| \log n)$ bits over point-to-point channels in the optimistic case, and an additional $\mathcal{O}(n^3 |C| \log n)$ bits over the broadcast channel in the pessimistic case (recall that this means that each party has to send or receive a total of $\mathcal{O}(n^4 |C| \log n)$ bits). In a nutshell, the protocol requires each party to perform $\mathcal{O}(1)$ VSSs in parallel for each multiplication gate in the circuit, and recall that in each VSS the dealer broadcasts $\mathcal{O}(n^2 \log n)$ bits. This is exactly the setting in which our parallel broadcast gives asymptotically free broadcast (Corollary 3.1.2). Thus, we get a protocol with

a total of $\mathcal{O}(n^4|C| \log n)$ bits (expected) and expected $\mathcal{O}(\text{depth}(C))$ rounds over point-to-point channels. Previously, using [82], this would have been resulted in expected $\mathcal{O}(n^6|C| \log n)$ communication complexity with $\mathcal{O}(\text{depth}(C))$ rounds.

Another line of work [80, 26, 76] in perfectly-secure MPC is based on the *player elimination* framework (introduced by Hirt and Maurer and Przydatek [80]). Those protocol identify parties that may misbehave and exclude them from the execution. Those protocols result in a total of $\mathcal{O}((n|C| + n^3) \log n)$ bits over point-to-point channels, and $\mathcal{O}(n \log n)$ bits over the broadcast channel. However, this comes at the expense of $\mathcal{O}(\text{depth}(C) + n)$ rounds. This can be compiled to $\mathcal{O}((n|C| + n^3) \log n)$ communication complexity with $\mathcal{O}(n^2 + \text{depth}(C))$ rounds using [53, 33], or to $\mathcal{O}((n|C| + n^7) \log n)$ communication complexity with $\mathcal{O}(n + \text{depth}(C))$ rounds (expected) using [82]. Using our broadcast, the communication complexity is $\mathcal{O}((n|C| + n^5) \log n)$ with $\mathcal{O}(n + \text{depth}(C))$ rounds (expected). We remark that in many setting, a factor n in round complexity should not be treated the same as communication complexity. Roundtrips are slow (e.g., 200ms delay for each roundtrip), whereas communication channels can send relatively large messages fast (1 or even 10Gbps).

On sequential and parallel composition of our broadcast. Like Feldman and Micali [66] and Katz and Koo [82] (and any $o(t)$ -round expected broadcast protocol), our protocol cannot provide simultaneous termination. Sequentially composing such protocols is discussed in Lindell, Lysyanskaya and Rabin [86], Katz and Koo [82] and Cohen et al. [54]. Regarding parallel composition, unlike the black-box parallel composition of broadcasts studied by Ben-Or and El-Yaniv [28], we rely on the idea of Fitzi and Garay [70] that applies to OLE-based protocols. The idea is that multiple broadcast sub-routines are run in parallel when only a single election per iteration is required for all these sub-routines. This reduces the overall cost and also guarantees that parallel broadcast is also constant expected number of rounds.

Modeling broadcast functionalities. We use standalone, simulation-based definition as in [41]. The standalone definition does not capture rounds in the ideal functionalities, or the fact that there is no simultaneous termination. The work of Cohen et al. [54] shows that one can simply treat the broadcast without simultaneous termination as an ideal broadcast as we provide (which, in particular, has simultaneous and deterministic termination). Moreover, it allows compiling a protocol using deterministic-termination hybrids (i.e., like our ideal functionalities) into a protocol that uses expected-constant-round protocols for emulating those hybrids (i.e., as our protocols) while preserving the expected round complexity of the protocol. We remark that in order to apply the compiler of [54], the functionalities need to follow a structure of (1) input from all parties; (2) leakage to the adversary; (3) output. For

simplicity, we did not write our functionalities using this specific format, but it is clear that our functionalities can be written in this style.

Our broadcast with strict-polynomial run time. Protocols in constant expected number of rounds might never terminate (although, with extremely small probability). Our protocols can be transformed into a protocol that runs in strict polynomial time using the approach of Goldreich and Petrank [75]: Specifically, after $\mathcal{O}(n)$ attempts to terminate, the parties can run the $\mathcal{O}(n)$ rounds protocol with guaranteed termination. See also [54].

3.1.3 Related Work

We review the related works below. Error-free byzantine agreement and broadcast are known to be possible only if $t < n/3$ holds [85, 94]. Moreover, Fischer and Lynch [69] showed a lower bound of $t + 1$ rounds for any deterministic byzantine agreement protocol or broadcast protocol. Faced with this barrier, Rabin [96] and Ben-Or [27] independently studied the effect of randomization on round complexity, which eventually culminated into the work of Feldman and Micali [68] who gave an expected constant round protocol for byzantine agreement with optimal resilience. Improving over this work, the protocol of [82] requires a communication of $\mathcal{O}(n^2L + n^6 \log n)$ for a message of size L bits, while achieving the advantage of expected constant rounds. In regards to the communication complexity, Dolev and Reischuk [65] established a lower bound of n^2 bits for deterministic broadcast or agreement on a single bit. With a round complexity of $\mathcal{O}(n)$, [53, 33] achieve a broadcast protocol with a communication complexity of $\mathcal{O}(n^2)$ bits.

We quickly recall the state of the art perfectly-secure broadcast extension protocols. Recall that these protocols aim to achieve the optimal complexity of $\mathcal{O}(nL)$ bits for sufficiently large message size L and utilize a protocol for bit broadcast. The protocol of [90, 72] communicates $\mathcal{O}(nL)$ bits over point-to-point channels and $\mathcal{O}(n^2)$ bits through a bit-broadcast protocol. The work of [89] improves the number of bits sent through a bit-broadcast protocol to $\mathcal{O}(n)$ bits. Both these extension protocols are constant round. The recent work of [47] presents a protocol that communicates $\mathcal{O}(nL + n^2 \log n)$ bits over point-to-point channels and a single bit through a bit-broadcast protocol. However, the round complexity of this protocol is $\mathcal{O}(n)$.

A few other works in different settings are given below. The notion of parallel broadcast was recently explored by Tsimos et al. [102] in the dishonest majority setting under cryptographic assumptions. Hirt and Zikas [79] studied the adaptive security of broadcast in the UC model, and improved the resilience of the ideal functionality to adaptive corruptions.

3.2 Technical Overview

We describe the high-level overview of our techniques. We start with our improved broadcast in Section 3.2.1, and then describe packed VSS in Section 3.2.2, followed by the gradecast protocol in Section 3.2.3. To aid readability, we summarize our different primitives and the relationship between them in Figure 3.1. In each one of the those primitives we improve over the previous works.

Primitive	P2P	Broadcast	Reference	Remarks
Broadcast	$\mathcal{O}(nL) + E(\mathcal{O}(n^4 \log n))$	–	Section 3.8.2	L bit message
Byzantine Agreement	$\mathcal{O}(n^2) + E(\mathcal{O}(n^4 \log n))$	–	Section 3.8.1	–
Gradecast	$\mathcal{O}(nL + n^3 \log n)$	–	Section 3.5	L bit message
Oblivious Leader Election	$\mathcal{O}(n^4 \log n)$	–	Section 3.7	–
Multi-moderated VSS	$\mathcal{O}(n^4 \log n)$	–	Section 3.6	Sharing $\mathcal{O}(n)$ values
Packed VSS (w. Gradecast)	$\mathcal{O}(n^3 \log n)$	–	Section 3.4	Sharing $\mathcal{O}(n)$ values
Packed VSS (w. Broadcast)	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(n^2 \log n)$	Section 3.4	Sharing $\mathcal{O}(n)$ values

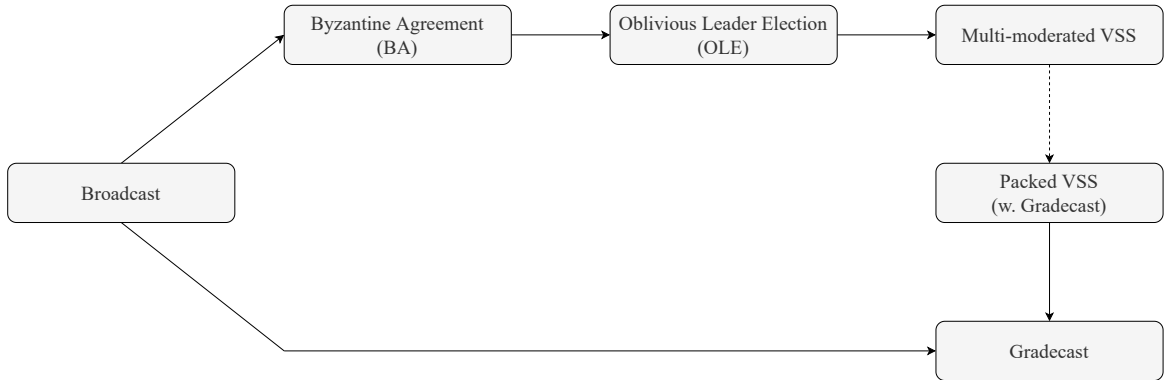


Figure 3.1: Roadmap of our building blocks. All lines are compositions, except for the line from Multi-moderated VSS to Packed VSS, which is a white-box modification.

3.2.1 Improved Broadcast in Constant Expected Rounds

Our starting point is a high-level overview of the broadcast protocol of Katz and Koo [82], which simplifies and improves the construction of Feldman and Micali [66]. Following the approach of Turpin and Coan [103] for broadcast extension closely, broadcast can be reduced to two primitives: Gradecast and Byzantine agreement.

1. **Gradecast:** A gradecast is a relaxation of broadcast, where a distinguished dealer transmits a message, and parties output the message together with a grade. If the dealer is honest, all honest parties are guaranteed to output the dealer’s message together with a grade 2. Moreover, if the dealer is corrupted and one honest party outputs grade 2,

then it is guaranteed that all other honest parties also output the same message, though maybe with a grade 1. Looking ahead, we show how to improve gradecast of message of length L bits from $\mathcal{O}(n^2L)$ bits to $\mathcal{O}(nL + n^3 \log n)$ bits, which is optimal for messages of $L = \Omega(n^2 \log n)$ bits. We overview our construction in Section 3.2.3.

2. **Byzantine agreement:** In Byzantine agreement all parties hold some bit as input, and all of them output a bit at the end of the protocol. If all honest parties hold the same value, then it is guaranteed that the output of all parties would be that value. Otherwise, it is guaranteed that the honest parties would agree and output the same (arbitrary) bit.

To implement broadcast, the dealer gradecasts its message M and then the parties run Byzantine agreement (BA) on the grade they received (using 1 as input when the grade of the gradecast is 2, and 0 otherwise). Then, if the output of the BA is 1, each party outputs the message it received from the gradecast, and otherwise it outputs \perp .

If the dealer is honest, then all honest parties receive grade 2 in the gradecast, and all would agree in the BA that the grade is 2. In that case, they all output M . If the dealer is corrupted, and all honest parties received grade 0 or 1 in the gradecast, they would all use 0 in the Byzantine agreement, and all would output \perp . The remaining case is when some honest parties receives grade 2 in the gradecast, and some receive 1. However, once there is a single honest party that received grade 2 in the gradecast, it is guaranteed that all honest parties hold the same message M . The Byzantine agreement can then go either way (causing all to output M or \perp), but agreement is guaranteed.

Oblivious leader election. It has been shown that to implement a Byzantine agreement (on a single bit), it suffices to obviously elect a leader, i.e., a random party among the parties. In a nutshell, a Byzantine agreement proceeds in iterations, where parties exchange the bits they believe that the output should be and try to see if there is an agreement on the output. When there is no clear indication of which bit should be the output, the parties try to see if there is an agreement on the output bit suggested by the elected leader. A corrupted leader might send different bits to different parties. However, once an honest leader is elected, it must have sent the same bit to all parties. In that case the protocol guarantees that all honest parties will agree in the next iteration on the output bit suggested by the leader, and halt.

Oblivious leader election is a protocol where the parties have no input, and the goal is to agree on a random value in $\{1, \dots, n\}$. It might have three different outcomes: (1) All parties agree on the same random index $j \in \{1, \dots, n\}$, and it also holds that P_j is honest; this is the preferable outcome; (2) All parties agree on the same index $i \in \{1, \dots, n\}$, but P_i is corrupted; (3) The parties do not agree on the index of the party elected. The goal is to

achieve the outcome (1) with constant probability, say $\geq 1/2$. Recall that once outcome (1) occurs then the Byzantine agreement succeeds. Achieving outcome (1) with constant number of rounds and with constant probability implies Byzantine agreement with constant expected number of rounds.

The key idea to elect a leader is to randomly choose, for each party, some random value c_i . Then, the parties choose an index j of the party for which c_j is minimal. To do that, we cannot let each party P_j choose its random value c_j , as corrupted parties would always choose small numbers to be elected. Thus, all parties contribute to the random value associated with each party. That is, each party P_k chooses $c_{k \rightarrow j} \in \{1, \dots, n^4\}$ as its share in the value c_j that will be assigned to P_j . Parties then define $c_j = \sum_{k=1}^n c_{k \rightarrow j} \bmod n^4$ as the random value associated with P_j . This guarantees that each value c_j is uniform.

However, just as in coin-tossing protocols, a party cannot publicly announce its random choices, since then it would allow a rushing adversary to choose its random values as a function of the announced values. This is prevented by using *verifiable secret sharing*. Verifiable secret sharing provides *hiding* – given t shares, it is impossible to determine what is the secret, and *binding* – at the end of the sharing phase, the dealer cannot change the secret, and reconstruction is guaranteed. The parties verifiably share their random values $c_{k \rightarrow j}$ for every k, j . After all parties share their values, it is safe to reconstruct the secret, reveal the random values, and elect the leader based on those values.

A problem: VSS uses a broadcast channel. A problem with the above solution is that protocols for VSS use a broadcast channel to reach an agreement on whether or not to accept the dealer’s shares. Yet, the good news is that broadcast is used only during the sharing phase. Replacing each broadcast with a gradedcast does not suffice since honest parties do not necessarily agree on the transmitted messages when corrupted senders gradedcast messages. This leads to the notion of “moderated VSS”, where the idea is to have a party that is responsible for all broadcasted message. Specifically, now there are two distinguished parties: a dealer P_k and a moderator P_j . The parties run the VSS where P_k is the dealer; whenever a participant has to broadcast a message m , it first gradedcasts it, and then the moderator P_j has to gradedcast the message it received. Each party can then compare between the two gradedcasted messages; however, the parties proceed the execution while using the message that the moderator had gradedcasted as the message that was broadcasted. At the end of the execution, each party outputs together with the shares, a grade for the moderator in $\{0, 1\}$. For instance, if the moderator ever gradedcasted some message and the message was received by some party P_i with grade ≤ 1 , then the grade that P_i gives the moderator is 0 — P_i cannot know whether other parties received the same message at all. The idea is that honest parties

might not necessarily output the same grade, but if there is one honest party that outputs grade 1, it is guaranteed that the VSS was successful, and we have binding. Moreover, if the moderator is honest, then all honest parties would give it grade 1.

Going back to leader election, the value $c_{k \rightarrow j}$ is distributed as follows: the parties run a VSS where P_k is the dealer and P_j is the moderator. After all values of all parties were shared (i.e., all parties committed to the values $c_{k \rightarrow j}$), each party defines for each moderator P_j the value $c_j = \sum_{k=1}^n c_{k \rightarrow j}$. If the grade of P_j was not 1 in all its executions as a moderator, then replace $c_j = \infty$. Each party elects the party P_ℓ for which c_ℓ is minimal.

If the moderator P_j is honest, then for both honest and corrupted dealer P_k , the VSS would end up with agreement, and all honest parties would give P_j grade 1 as a moderator. The value $c_j = \sum_{k=1}^n c_{k \rightarrow j} \bmod n^4$ would be the same for all honest parties, and it must distribute uniformly as honest dealers contributed random values in this sum. Likewise, if a moderator P_j is corrupted but some honest party outputs grade 1 in all executions where P_j served as a moderator, then the value $c_j = \sum_{k=1}^n c_{k \rightarrow j} \bmod n^4$ must be the same for all honest parties, and it also must be random, as honest dealers contributed random values. There might be no agreement if some honest parties gave grade 1 for that moderator, while others did not and defined $c_j = \infty$. In that case, we might not have an agreement on the elected leader. However, it is guaranteed that the value c_j is distributed uniformly. Thus, the inconsistency is bounded with constant probability (roughly $t/n \leq 1/3$).

Our improvements. As noticed above, each party participates as the dealer in n executions, and as the role of the moderator in n executions. Thus, we have a total of n^2 executions of VSS. First, we show a new protocol that enables a dealer to pack $\mathcal{O}(n)$ secrets at the cost of just one VSS (assuming broadcast), called packed VSS (see an overview in Section 3.2.2). For leader election, we have to replace the broadcast in the packed VSS with a gradecast (with a moderator).

However, we cannot just pack all the $\mathcal{O}(n)$ values $c_{k \rightarrow j}$ where P_k is the dealer in one instance of a VSS with a moderator since each one of the secrets corresponds to a different moderator. We, therefore, introduce a new primitive which is called “Multi-moderated packed secret sharing”: The dealer distributes $\mathcal{O}(n)$ values, where each corresponds to a different moderator, and have all parties serve as moderator in one shared execution of a VSS.

More precisely, the packed VSS uses several invocations of broadcasts in the sharing phase, just as a regular VSS. Until the very last round, the dealer also serves as the moderator within each of those broadcasts. In the last round, there is a vote among the parties whether accept or reject the dealer, where the vote is supposed to be performed over the broadcast

channel. At this point, the execution is forked to $\mathcal{O}(n)$ executions. Each corresponds to a different moderator, where the moderator moderates just the last round’s broadcasts. The idea is that the vast majority of the computation is shared between all $\mathcal{O}(n)$ executions, thus the additional cost introduced for each moderator is small. This allows us to replace all n executions where P_i serves as a dealer with just one execution where P_i is the dealer and other $\mathcal{O}(n)$ parties are moderators at the same time.

Another obstacle worth mentioning is that within multi-moderated packed VSS, the dealer broadcasts $\mathcal{O}(n^2 \log n)$ bits, whereas other participant broadcasts at most $\mathcal{O}(n \log n)$ bits. Our gradecast is not optimal for this message size, and thus when replacing those broadcasts with gradecasts, the overall cost would be $\mathcal{O}(n^5 \log n)$. We can do better by considering all the multi-moderated VSSs in parallel. Each party then participates in $\mathcal{O}(1)$ executions as a dealer and in $\mathcal{O}(n)$ executions as a participant. Therefore, each party has to broadcast $\mathcal{O}(n^2 \log n)$ bits in all invocations of multi-moderated packed VSS combined ($\mathcal{O}(n^2 \log n)$ bits when it serves as a dealer, and $(n - 1) \times \mathcal{O}(n \log n)$ when it serves as a participant). For that size of messages, our gradecast is optimal.

To conclude, to obtain our broadcast, we build upon [66, 82] and introduce: (1) an optimal gradecast protocol for $\Omega(n^2 \log n)$ messages which is used twice – for gradecasting the message before running the Byzantine agreement and within the Byzantine agreement as part of the VSSs; (2) a novel multi-moderated packed secret sharing, which is based on a novel packed VSS protocol; (3) carefully combine all the $\mathcal{O}(n)$ invocations of multi-moderated packed secret sharing to amortize the costs of the gradecasts.

When comparing to the starting point of $\mathcal{O}(n^2 L)$ plus $E(\mathcal{O}(n^6 \log n))$ of [82], the improved gradecast allows us to reduce the first term to $\mathcal{O}(nL)$, for large enough messages. Regarding the second term, packing $\mathcal{O}(n)$ values in the VSS reduces one n factor, and the improved gradecast within the VSS reduces another n factor. Overall this brings us to $\mathcal{O}(nL)$ plus $E(\mathcal{O}(n^4 \log n))$.

3.2.2 Packed Verifiable Secret Sharing

Our packed verifiable secret sharing protocol is the basis of the multi-moderated VSS. We believe that it will find applications in future constructions of MPC protocols, and is of independent interest. Communication cost wise, the best-known constant-round perfect VSS sharing one secret is $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels in the optimistic case, and additional $\mathcal{O}(n^2 \log n)$ bits over the broadcast channel in the pessimistic case [29, 74, 18]. Here, we retain the same cost, yet “pack” $t + 1$ secrets in one bivariate polynomial and generate $t + 1$ independent Shamir-sharings at one go. We remark that in asynchronous setting,

with the optimal threshold of $t < n/4$, this goal has been achieved in [50, 51].

Sharing more secrets at one go. Our goal is to generate Shamir-sharing of $t + 1$ secrets, s_{-t}, \dots, s_0 , at once. Denoting Shamir-sharing of a secret s by $\langle s \rangle$, our goal is to produce $\langle s_{-t} \rangle, \dots, \langle s_0 \rangle$ using a single instance of a VSS. For this, the dealer chooses a degree- $(2t, t)$ bivariate polynomial¹ $S(x, y)$ such that $S(l, 0) = s_l$ for each $l \in \{-t, \dots, 0\}$. We set $f_i(x) = S(x, i)$ of degree $2t$ and $g_i(y) = S(i, y)$ of degree- t and observe that for every i, j it holds that $f_i(j) = S(j, i) = g_j(i)$. The goal of the verification part is that each P_i will hold $f_i(x)$ and $g_i(y)$ on the same bivariate polynomial $S(x, y)$. Then, each degree- t univariate polynomial $g_l(y)$ for $l \in \{-t, \dots, 0\}$ is the standard Shamir-sharing of s_l amongst the parties. Once the shares of the parties are consistent, each party P_i can locally compute its share on $g_l(y)$ as $g_l(i) = f_i(l)$.

Our protocol is a strict improvement of [6]. Specifically, the work of [6] considers the VSS protocol of [29] when the dealer uses a $(2t, t)$ -polynomial instead of a degree- (t, t) polynomial. It observes that by minor modifications, the protocol still provides weak verifiability even though the sharing is done on a higher degree polynomial. By “weak”, we mean that the reconstruction phase of the polynomial might fail in the case of a corrupted dealer. Nevertheless, the guarantee is that the reconstruction phase would either end up successfully reconstructing $S(x, y)$, or \perp , and whether it would succeed or not depends on the adversary’s behavior. In contrast, in a regular (“strong”) VSS, reconstruction is always guaranteed.

The work of [6] utilizes this primitive to improve the efficiency of the degree-reduction step of the BGW protocol. However, this primitive is weak and does not suffice for most applications of VSS. For instance, it cannot be used as a part of our leader election protocol: The adversary can decide whether the polynomial would be reconstructed or not. Thus there is no “binding”, and it can choose, adaptively and based on the revealed secrets of the honest parties, whether the reconstruction would be to the secret values or some default values. As such, it can increase its chance of being elected.

Our work: achieving strong binding. In our work, we show how to achieve strong binding. We omit the details in this high-level overview of achieving weak verifiability of [6] secret sharing while pointing out that the protocol is a variant of the VSS protocol of [29]. For our discussion, the protocol reaches the following stage: If the dealer is not discarded, then there is a CORE of $2t + 1$ parties that hold shares of a unique bivariate polynomial $S(x, y)$, and this set of parties is public and known to all (it is determined based on votes performed over the broadcast channel). Each party P_i in CORE holds two univariate shares $f_i(x) = S(x, i)$

¹We call a bivariate polynomial where the degree in x is $2t$ and in y is t , i.e., $S(x, y) = \sum_{i=0}^{2t} \sum_{j=0}^t a_{i,j} x^i y^j$ as a $(2t, t)$ -bivariate polynomial.

of degree- $2t$ and $g_i(y) = S(i, y)$ of degree- t . Each party P_j for $j \notin \text{CORE}$ holds a polynomial $g_j(y) = S(j, y)$, where some of those polynomials are also public and were broadcasted by the dealer. In case the dealer is honest, then all honest parties are part of CORE, whereas if the dealer is corrupted, then it might be that only $t + 1$ honest parties are part of CORE. To achieve strong binding, the dealer has to provide shares for parties outside CORE, publicly, and in a constant number of rounds.

The first step is to make all the polynomials $g_j(y)$ for each $j \notin \text{CORE}$ public. This is easy, since each such polynomial is of degree t . The dealer can broadcast it, and the parties in CORE vote whether to accept. If there are no $2t + 1$ votes to accept, then the dealer is discarded. Since the shares of the honest parties in CORE are consistent and define a unique $(2t, t)$ -bivariate polynomial $S(x, y)$, the dealer cannot publish any polynomial $g_j(y)$ which is not $S(j, y)$. Any polynomial $g'_j(y) \neq S(j, y)$ can agree with at most t points with $S(j, y)$ and thus it would receive at most t votes of honest parties in CORE, i.e., it cannot reach $2t + 1$ votes.

The next step is to make the dealer also publicize the shares $f_j(x)$ for each $j \notin \text{CORE}$. This is more challenging since each $f_j(x)$ is of degree- $2t$, and therefore achieving $2t + 1$ votes is not enough, as t votes might be false. Therefore, the verification is more delicate:

1. First, the parties in CORE have to vote OK on the f -polynomials that the dealer publishes. If there are less than $2t + 1$ votes, the dealer is discarded.
2. Second, for each party P_j in CORE that did not vote OK, the dealer is required to publish its $g_j(y)$ polynomial. The parties in CORE then vote on the revealed polynomials as in the first step of boosting from weak to strong verification.

To see why this works, assume that the dealer tries to distribute a polynomial $f'_j(x) \neq S(x, j)$. Then, there must exist an honest party such that its share does not agree with $f'_j(x)$. If $f'_j(x)$ does not agree with shares that are public, then it would be immediately discarded. If $f'_j(x)$ does not agree with a share of an honest party P_k that is part of CORE, then $g_k(y)$ would become public in the next round, and the dealer would be publicly accused. The dealer cannot provide a share $g_k(y) \neq S(k, y)$ for the same reason as the first step of boosting from weak to strong VSS. At the end of this step we have that all honest parties are either part of CORE and their shares are private, or they are not in CORE and their shares are public. Overall, all honest parties hold shares on the bivariate polynomial $S(x, y)$. We refer to section 3.4 for the formal protocol description.

3.2.3 Optimal Gradecast

A crucial building block in our construction is gradecast. We show how to implement gradecast of a message of length L bits using total communication of $\mathcal{O}(n^3 \log n + nL)$ bits. For this overview, we just deal with the case where the dealer is honest and show that all honest parties output the message that the dealer gradecasted with grade 2. We leave the case of a corrupted dealer to the relevant section (Section 3.5).

Data dissemination. Our construction is inspired in part by the data dissemination protocol of [61], while we focus here on the synchronous settings. In the task of data dissemination, $t + 1$ honest parties hold as input the same input M , while other honest parties hold the input \perp , and the goal is that all honest parties receive the same output M in the presence of t corrupted parties. In our protocol, assume for simplicity messages of size $(t + 1)^2$ field elements (i.e., a degree- (t, t) bivariate polynomial). Data dissemination can be achieved quite easily: (1) Each honest party sends to each party P_j the univariate polynomials $S(x, j), S(j, y)$. (2) Once a party receives $t + 1$ messages with the same pair of univariate polynomials, it forwards those polynomials to all others. An adversary might send different polynomials, but it can never reach plurality $t + 1$. (3) After all the honest parties forwarded their polynomials to the others, we are guaranteed that each party holds $2t + 1$ correct shares of S and at most t incorrect shares. Each party can reconstruct S efficiently using Reed Solomon decoding. Note that this procedure requires the transmission of $\mathcal{O}(n^3 \log n)$ bits overall. Therefore, our goal in the gradecast protocol is to reach a state where $t + 1$ honest parties hold shares of the same bivariate polynomial.

Gradecast. For the sake of exposition, we first describe a simpler protocol where the dealer is computationally unbounded, and then describe how to make the dealer efficient. Again, assume that the input message of the dealer is encoded as a bivariate polynomial $S(x, y)$. The dealer sends the entire bivariate polynomial to each party. Then, every pair P_i and P_j exchange the polynomials $S(x, i), S(i, y), S(x, j), S(j, y)$. The two parties check whether they agree on those polynomials or not. If P_i sees that the polynomials it received from P_j are the same as it received from the dealer, then it adds j to a set Agreed_i . The parties then send their sets Agreed_i to the dealer, who defines an undirected graph where the nodes are the set $\{1, \dots, n\}$ and an edge $\{i, j\}$ exists if and only if $i \in \text{Agreed}_j$ and $j \in \text{Agreed}_i$. The dealer then (inefficiently) finds a maximal clique $K \subseteq \{1, \dots, n\}$ of at least $2t + 1$ parties and gradecasts K to all parties using a naïve gradecast protocol of [66, 82] (note that this is a gradecast of case $\mathcal{O}(n^2 L)$ with $L = \mathcal{O}(n \log n)$). A party P_i is happy if: (1) $i \in K$; (2) it received the gradecast message of the dealer with grade 2; and (3) $K \subseteq \text{Agreed}_i$. The parties then proceed

to data dissemination protocol.

The claim is that if the dealer is honest, then at least $t + 1$ honest parties are happy, and they all hold the same bivariate polynomial. This is because the set of honest parties defines a clique of size $2t + 1$, and any clique that the honest dealer finds of cardinality $2t + 1$ must include at least $t + 1$ honest parties. The result of the data dissemination protocol is that all honest parties output S . If the dealer is corrupted, we first claim that all honest parties that are happy must hold the same bivariate polynomial. Any two honest parties that are happy must be part of the same clique K that contains at least $t + 1$ honest parties, and all honest parties in that clique must agree with each other (all see the same clique K defined by the dealer, and verified that they agreed with each other). The univariate polynomials exchanged between those $t + 1$ honest parties define a unique bivariate polynomial. Again, data dissemination would guarantee that all honest parties would output that bivariate polynomial.

On making the dealer efficient. To make the dealer efficient, we rely on a procedure that finds an approximation of a clique, known as the STAR technique, introduced by [39]. In the technical section, we show how we can use this approximation of a clique, initially introduced for the case of $t < n/4$, to the much more challenging scenario of $t < n/3$. We refer to Section 3.5 for the technical details.

Organization. The rest of the chapter is organized as follows. In Section 3.3 we provide preliminaries and notations. In Section 3.4 we describe our packed verifiable secret sharing, followed by our gradecast in Section 3.5. We then proceed to multi-moderated packed secret sharing (Section 3.6), oblivious leader election (Section 3.7) and we conclude with our broadcast protocol in Section 3.8.

3.3 Preliminaries

3.3.1 Network Model and Notations

We consider a synchronous network model where the parties in $\mathcal{P} = \{P_1, \dots, P_n\}$ are connected via pairwise private and authenticated channels. Additionally, for some of our protocols we assume the availability of a broadcast channel, which allows a party to send an identical message to all the parties. One of the goals of this work is to implement such a broadcast channel over the pairwise private channels, and we mention explicitly for each protocol whether a broadcast channel is available or not. The detailed description of the network model and security definition is discussed in Section 2.

Our protocols are defined over a finite field \mathbb{F} where $|\mathbb{F}| > n + t + 1$. We consider two sets

of n and $t + 1$ distinct elements from \mathbb{F} publicly known to all the parties, which we denote by $\{1, \dots, n\}$ and $\{-t, \dots, 0\}$ respectively. We use $\langle v \rangle$ to denote the degree- t Shamir-sharing of a value v among parties in \mathcal{P} .

3.3.2 Bivariate Polynomials

As described earlier, a degree (l, m) -bivariate polynomial over \mathbb{F} is of the form $S(x, y) = \sum_{i=0}^l \sum_{j=0}^m b_{ij} x^i y^j$ where $b_{ij} \in \mathbb{F}$. The polynomials $f_i(x) = S(x, i)$ and $g_i(y) = S(i, y)$ are called i^{th} f and g univariate polynomials of $S(x, y)$ respectively. In our protocol, we use $(2t, t)$ -bivariate polynomials where the i^{th} f and g univariate polynomials are associated with party P_i for every $P_i \in \mathcal{P}$.

Claim 3.3.1 ([6, Claim 3.4]). *Let t be a non-negative integer, $\{1, \dots, t + 1\}$ be distinct elements in \mathbb{F} and $f_1(x), \dots, f_{t+1}(x)$ be $t + 1$ univariate polynomials of degree at most $2t$. Then, there exists a unique $(2t, t)$ -bivariate polynomial $S(x, y)$ such that $S(x, i) = f_i(x)$ holds for every $i \in [t + 1]$.*

Claim 3.3.2 ([11, Lemma 2.7]). *Let $C \subseteq D \subseteq \{1, \dots, n\}$ be two sets such that $|C| \geq t + 1$ and $|D| \geq 2t + 1$. Let $\{f_i(x)\}_{i \in C}$ be a set of degree- $2t$ polynomials and $\{g_j(y)\}_{j \in D}$ be a set of degree- t polynomials over \mathbb{F} . If for every $i \in C$ and every $j \in D$ it holds that $f_i(j) = g_j(i)$, then there exists a unique $(2t, t)$ -bivariate polynomial $S(x, y)$ such that $S(x, i) = f_i(x)$ holds for every $i \in C$ and $S(j, y) = g_j(y)$ holds for every $j \in D$.*

Claim 3.3.3 ([6, Claim 3.6]). *Let $\mathcal{C} \subset \{1, \dots, n\}$ be a set such that $|\mathcal{C}| \leq t$ and let $p(x)$ and $q(x)$ be two degree- $2t$ polynomials such that $p(i) = q(i)$ holds for every $i \in \mathcal{C}$. Then, the probability distributions $\{(i, S_p(x, i), S_p(i, y))\}_{i \in \mathcal{C}}$ and $\{(i, S_q(x, i), S_q(i, y))\}_{i \in \mathcal{C}}$ are identical, where $S_p(x, y)$ and $S_q(x, y)$ are $(2t, t)$ -bivariate polynomials chosen under the constraint that $S_p(x, 0) = p(x)$ and $S_q(x, 0) = q(x)$ respectively.*

3.3.3 Finding (n, t) -STAR

Definition 3.3.4. *Let G be a graph over the nodes $\{1, \dots, n\}$. We say that a pair (C, D) of sets such that $C \subseteq D \subseteq \{1, \dots, n\}$ is an (n, t) -star in G if the following hold:*

- $|C| \geq n - 2t$,
- $|D| \geq n - t$,
- For every $j \in C$ and every $k \in D$, the edge (j, k) exists in G .

Canetti [39] showed that if a graph has a clique of size $n - t$, then there exists an efficient algorithm which always finds an (n, t) -star. For completeness, we describe the algorithm for

finding an (n, t) -star in Algorithm 3.3.5 which is taken verbatim from [30, 40]. We describe the (n, t) -star finding algorithm [30, 40] below.

Algorithm 3.3.5: STAR – efficiently finding a star

Input: an undirected graph G (over the nodes $\{1, \dots, n\}$), a parameter t .

1. Find a maximum matching M in \overline{G} . Let N be the set of matched nodes (namely, the endpoints of the edges in M) and let $\overline{N} := \{1, \dots, n\} \setminus N$.
2. Let T be the set of triangle-heads, i.e., all vertices that are not endpoints of the matching but they have two neighbors in the matching.

$$T := \{i \in \overline{N} \mid \exists j, k \text{ s.t. } (j, k) \in M \text{ and } (i, j), (i, k) \in \overline{G}\} .$$

Let $C := \overline{N} \setminus T$.

3. Let B be the set of matched nodes that have neighbors in C . That is, set:

$$B := \{j \in N \mid \exists i \in C \text{ s.t. } (i, j) \in \overline{G}\} .$$

Let $D := \{1, \dots, n\} \setminus B$.

4. **Output:** If $|C| \geq n - 2t$ and $D \geq n - t$ then output (C, D) . Otherwise, output “star not found”.

It was shown in [40, 30] that if a graph has a clique of size $n - t$, then the above procedure halts with a (C, D) star.

Claim 3.3.6. *Let G be a graph over $\{1, \dots, n\}$ such that if P_i and P_j are honest then $\{i, j\} \in G$. Then, C contains at least $t + 1$ indices of honest parties.*

Proof. Since honest parties trust each other, we have a clique of size at least $2t + 1$ in G and thus a (C, D) -star will be found. Since there are always edges between two honest parties in G , all the edges in \overline{G} are either between an honest party and a corrupted party, or between a pair of corrupted parties. Let x be the number of edges in the matching that are between pairs of corrupted parties, and let y be the number of edges in the matching that are between an honest party and a corrupted party. We have that $x + y \leq t$. Next, we claim that the number of honest parties in T (i.e., triangle-heads) is bounded by x . The only triangles in questions are those between an honest party as a head and the two neighbors as corrupted parties that are also in the matching. We claim that each edge in the matching between a pair

of corrupted parties can be a part of only one triangle. That is, for $(j, k) \in M$ there exists at most one honest $i \in \overline{N}$ for which $i \in T$. Otherwise, if there exists a pair $i_1, i_2 \in \overline{N}$ such that $i_1, i_2 \in T$, then we can find a larger matching: instead of taking $(j, k) \in M$ we would take (i_1, j) and (i_2, k) , in contradiction to the maximality of M .

To conclude, C is defined as $(\{1, \dots, n\} \setminus N) \setminus T$. In N there are y indices of honest parties, and in T at most x indices. Since $x + y \leq t$, we obtain that C contains at least $t + 1$ indices of honest parties. \square

3.4 Packed Verifiable Secret Sharing

Here we present a packed VSS to generate Shamir sharing of $t + 1$ secrets at the cost of $\mathcal{O}(n^2 \log n)$ bits point-to-point and broadcast communication.

The Functionality. On holding $t + 1$ secrets s_{-t}, \dots, s_0 , the dealer chooses a uniformly random $(2t, t)$ -bivariate polynomial $S(x, y)$ such that $S(l, 0) = s_l$ for each $l \in \{-t, \dots, 0\}$ and uses the polynomial as its input. Our functionality for VSS is as follows, followed by the VSS protocol.

Functionality 3.4.1: \mathcal{F}_{VSS} – Packed Verifiable Secret Sharing Functionality

Input: The dealer holds a polynomial $S(x, y)$.

1. The dealer sends $S(x, y)$ to the functionality.
 2. If $S(x, y)$ is of degree at most $2t$ in x and at most t in y , then the functionality sends to each party P_i the two univariate polynomials $S(x, i), S(i, y)$. Otherwise, the functionality sends \perp to all parties.
-

Protocol 3.4.2: Π_{pVSS} – Packed VSS Protocol

Common input: The description of a field \mathbb{F} , two sets of distinct elements from it denoted as $\{1, \dots, n\}$ and $\{-t, \dots, 0\}$.

Input: The dealer holds a bivariate polynomial $S(x, y)$ of degree at most $2t$ in x and at most t in y . Each P_i initialises a happy bit $\text{happy}_i = 1$ ¹.

1. **(Sharing)** The dealer sends $(f_i(x), g_i(y))$ to P_i where $f_i(x) = S(x, i), g_i(y) = S(i, y)$.

¹The happy bits will be used later for Multi-Moderated VSS in Section 3.6.

2. **(Pairwise Consistency Checks)** Each P_i sends $(f_i(j), g_i(j))$ to every P_j . Let (f_{ji}, g_{ji}) be the values received by P_i from P_j . If $f_{ji} \neq g_i(j)$ or $g_{ji} \neq f_i(j)$, P_i broadcasts $\text{complaint}(i, j, f_i(j), g_i(j))$.
 3. **(Conflict Resolution)** For each $\text{complaint}(i, j, u, v)$ such that $u \neq S(j, i)$ or $v \neq S(i, j)$, dealer broadcasts $g_i^D(y) = S(i, y)$. Let pubR be the set of parties for which the dealer broadcasts $g_i^D(y)$. Each $P_i \in \text{pubR}$ sets $\text{happy}_i = 0$. For two mutual complaints $(\text{complaint}(i, j, u, v), \text{complaint}(j, i, u', v'))$ with either $u \neq u'$ or $v \neq v'$, if the dealer does not broadcast anything, then discard the dealer.
 4. **(Identifying the CORE Set)** Each $P_i \notin \text{pubR}$ broadcasts OK if $f_i(k) = g_k^D(i)$ holds for every $k \in \text{pubR}$. Otherwise, P_i sets $\text{happy}_i = 0$. Let CORE be the set of parties who broadcasted OK. If $|\text{CORE}| < 2t + 1$, then discard the dealer.
 5. **(Revealing f -polynomials for non-CORE parties)** For each $P_k \notin \text{CORE}$, the dealer broadcasts $f_k^D(x) = S(x, k)$. Discard the dealer if for any $P_j \in \text{pubR}$ and $P_k \notin \text{CORE}$, $g_j^D(k) \neq f_k^D(j)$. Each $P_i \notin \text{pubR}$ broadcasts OK if $f_k^D(i) = g_i(k)$ holds for every broadcasted $f_k^D(x)$. Otherwise P_i sets $\text{happy}_i = 0$. Let $K = \{P_j | P_j \notin \text{pubR} \text{ and did not broadcast OK}\}$.
 6. **(Opening g -polynomials for complaining parties)** For each $P_j \in K$, the dealer broadcasts $g_j^D(y) = S(j, y)$. Set $\text{pubR} = \text{pubR} \cup K$. Discard the dealer if $f_k^D(j) \neq g_j^D(k)$ for any $P_k \notin \text{CORE}$ and $P_j \in K$. Each $P_i \in \text{CORE}$ with $\text{happy}_i = 1$ broadcasts OK if $f_i(j) = g_j^D(i)$ for every broadcasted $g_j^D(y)$. Otherwise, P_i sets $\text{happy}_i = 0$. If at least $2t + 1$ parties do not broadcast OK, then discard the dealer.
 7. **(Output)** If the dealer is discarded, then each P_i outputs \perp . Otherwise, P_i outputs $(f_i(x), g_i(y))$, where $f_i(x) = f_i^D(x)$ if $P_i \notin \text{CORE}$ and $g_i(y) = g_i^D(y)$ if $P_i \in \text{pubR}$.
-

Theorem 3.4.3. *Protocol Π_{pVSS} (Protocol 3.4.2) securely realizes \mathcal{F}_{VSS} (Functionality 3.4.1) in the presence of a static malicious adversary controlling up to t parties with $t < n/3$.*

Proof. Let \mathcal{A} be an adversary in the real world. We show the existence of a simulator \mathcal{SJM} in the ideal world, such that for any set of corrupted parties \mathcal{C} and for all inputs, the output of all parties in the real protocol with \mathcal{A} is identical to the output in the ideal world with \mathcal{SJM} . Depending on whether the dealer is honest or not, we have the following two cases.

Case 1 - The dealer is honest. In this case, the dealer always holds a valid $(2t, t)$ -bivariate polynomial $S(x, y)$. The simulator proceeds as follows:

1. SJM invokes \mathcal{A} on auxiliary input z .
2. SJM receives from \mathcal{F}_{VSS} , the polynomials $f_i(x), g_i(y)$ for every $P_i \in \mathcal{C}$ and simulates the protocol execution for \mathcal{A} :
 - (a) **Sharing:** SJM sends $(f_i(x), g_i(y))$ to \mathcal{A} for every $P_i \in \mathcal{C}$ on behalf of the dealer.
 - (b) **Pairwise Consistency Checks:** SJM sends $(g_i(j), f_i(j))$ to \mathcal{A} for every $P_i \in \mathcal{C}$ and every honest P_j . SJM receives from \mathcal{A} the values (f_{ij}, g_{ij}) for each honest party P_j and every $P_i \in \mathcal{C}$. If $f_{ij} \neq f_i(j)$ or $g_{ij} \neq g_i(j)$, SJM broadcasts $\text{complaint}(j, i, g_i(j), f_i(j))$ on behalf of P_j . SJM also receives $\text{complaint}(\cdot, \cdot, \cdot, \cdot)$ broadcasted by \mathcal{A} .
 - (c) **Conflict Resolution:** The dealer never broadcasts $g_j^D(y)$ for honest parties. For every $\text{complaint}(i, j, u, v)$ from \mathcal{A} , SJM checks if $u = f_i(j)$ and $v = g_i(j)$. If not, SJM broadcasts $g_i^D(y)$ on behalf of the dealer. Define pubR to be the set of parties for which $g_i^D(y)$ was broadcasted.
 - (d) **CORE Set Identification:** An honest party never belongs to pubR . Since the dealer is honest, $f_j(i) = g_i^D(j)$ holds for every honest P_j and every $P_i \in \text{pubR}$. The dealer broadcasts OK on behalf of every honest party and receives the OK messages broadcasted by \mathcal{A} . Define CORE to be the set of parties who broadcasted OK.
 - (e) **Revealing f -polynomials non-CORE parties:** An honest party will always be a part of CORE . On behalf of the dealer, SJM reveals $f_i^D(x)$ for each $P_i \notin \text{CORE}$. These polynomials will always be consistent with the honest parties' polynomials and the g -polynomials revealed publicly. SJM broadcasts OK on behalf of each honest party and receives the OK broadcasted by \mathcal{A} . Let K be the parties which did not broadcast OK.
 - (f) **Opening g -polynomials for parties in K :** Let $\text{pubR} = \text{pubR} \cup K$. On behalf of the dealer, SJM reveals $g_i^D(y)$ for each $P_i \in K$. These polynomials will always be consistent with the f -polynomials of honest parties and the f -polynomials revealed publicly. SJM broadcasts OK on behalf of each honest party and receives the OK messages broadcasted by \mathcal{A} .
3. **Output:** SJM outputs whatever \mathcal{A} outputs, and halts.

It can be observed that, since the protocol as well as the simulation is deterministic, the

adversary's view in the real execution and ideal execution is identical. Hence, our goal is to now show that the output of honest parties is the same in the real and ideal executions.

In the ideal execution, an honest dealer always invokes the functionality with a valid $(2t, t)$ -bivariate polynomial $S(x, y)$. Thus, each honest party P_i outputs the polynomials $f_i(x) = S(x, i)$ and $g_i(y) = S(i, y)$ which it receives from the functionality, and never outputs \perp . Moreover, the corrupted parties do not have inputs and hence do not influence the output of the honest parties. We will show that the same holds in the real execution as well.

In the real execution, since the dealer is honest, it always holds a valid $(2t, t)$ -bivariate polynomial $S(x, y)$ and sends $f_i(x)$ and $g_i(y)$ as prescribed by the protocol to every P_i . As per the protocol specification, a party's g and f polynomials do not change unless they are revealed publicly by the dealer in Step 3, 5 or 6. However, an honest dealer never reveals an honest party's polynomials during these phases. Hence, during the output phase, an honest party either outputs $f_i(x), g_i(y)$ consistent with $S(x, y)$, or \perp . We thus proceed to show that an honest party never outputs \perp .

Recall that an honest party outputs $f_i(x)$ and $g_i(y)$ if and only if at least $2t + 1$ parties with $\text{happy} = 1$ broadcast OK during Step 6. Thus, it suffices to show that all the honest parties have their happy bit as 1 and broadcast OK. An honest party P_i has $\text{happy}_i = 1$ and broadcasts OK during Step 6 if and only if the following conditions hold:

1. While resolving complaints, the dealer never broadcasts $g_i^D(y)$.
2. The dealer resolves all pairs of complaints of the type $\text{complaint}(j, k, u, v)$ and $\text{complaint}(k, j, u', v')$ where $u \neq u'$ or $v \neq v'$.
3. All $g_k^D(y)$ broadcasted by the dealer in Step 3 satisfy $f_i(k) = g_k^D(i)$.
4. CORE set includes at least $2t + 1$ parties.
5. All $f_j^D(x)$ broadcasted by the dealer for every $P_j \notin \text{CORE}$ in Step 5 and $g_k^D(y)$ broadcasted for every $P_k \in \text{pubR}$ in Step 3 satisfy $f_j^D(k) = g_k^D(j)$, and $f_j^D(i) = g_i(j)$.
6. All $g_k^D(y)$ broadcasted by the dealer for every $P_k \in K$ in Step 6 and all $f_j^D(x)$ broadcasted for every $P_j \notin \text{CORE}$ in Step 5 satisfy $f_j^D(k) = g_k^D(j)$, and $f_i(k) = g_k^D(i)$.

Therefore we conclude that in the real execution, every honest party has $\text{happy}_i = 1$ broadcasts OK in Step 6 and hence every honest party P_i outputs $f_i(x)$ and $g_i(y)$ identical to the ideal execution.

Case 2 - The dealer is corrupt. In this case, the adversary \mathcal{A} controls the dealer. The honest parties do not have any input to the protocol and the protocol is deterministic. The simulator proceeds as follows:

1. \mathcal{SJM} invokes \mathcal{A} on auxiliary input z .
2. \mathcal{SJM} plays the role of all the honest parties while interacting with \mathcal{A} , as specified by protocol Π_{pVSS} (Protocol 3.4.2).
3. Let P_j be an arbitrary honest party emulated by \mathcal{SJM} . From its output in the simulated execution, let \mathcal{G} be the set of parties that broadcasted OK in the simulation in Step 6. Then,
 - (a) If $|\mathcal{G}| \geq 2t+1$, then let $\mathcal{H} \subset \mathcal{G} \setminus \mathcal{C}$ be the set of $t+1$ honest parties which broadcasted OK. \mathcal{SJM} finds the unique $(2t, t)$ -bivariate polynomial, say $S(x, y)$ that satisfies $f_i(x) = S(x, i)$ for every $P_i \in \mathcal{H}$. Such a polynomial always exists by virtue of Claim 3.3.1. \mathcal{SJM} sends $S(x, y)$ to \mathcal{F}_{VSS} to allow the honest parties to learn their output, and receives the output $f_i(x), g_i(y)$ for each $P_i \in \mathcal{C}$.
 - (b) Otherwise, \mathcal{SJM} invokes \mathcal{F}_{VSS} with an invalid polynomial, say $S(x, y) = x^{2t+1}$ causing all the honest parties to receive \perp in the ideal execution.
4. \mathcal{SJM} outputs whatever \mathcal{A} outputs, and halts.

Since the simulator emulates the honest parties as in the real execution of the protocol, the view of the adversary in the real and ideal world is identical. Thus, it remains to be shown that the output of the honest parties in the ideal world is the same as that in the real execution. For this, we consider the following two cases:

Case I - There exists an honest party that outputs \perp in the real execution. In such a case, we claim that all the honest parties output \perp . An honest party outputs \perp only if (i) the dealer does not resolve all mutual complaints (ii) CORE set (decided based on OK messages in Step 4) includes less than $2t + 1$ parties, (iii) any of the verification checks on the publicly revealed polynomials fail, or (iii) less than $2t + 1$ parties from CORE broadcast OK in Step 6. In all of the above cases, the corresponding messages are broadcasted, and hence all honest parties output \perp . Since the real execution and simulated executions are identical, all the simulated honest parties will output \perp . In this case, the simulator invokes the functionality with $S(x, y) = x^{2t+1}$, which in turn rejects the polynomial and sends \perp to all the honest parties.

Case II - No honest party outputs \perp in the real execution. In this case, we want to show that each honest party P_i holds $f_i(x) = S(x, i)$ and $g_i(y) = S(i, y)$ consistent with some unique $(2t, t)$ -bivariate polynomial $S(x, y)$.

Observe that, if an honest party did not output \perp , it implies that at least $2t + 1$ parties

from CORE broadcast OK in Step 6. This in turn implies that there exists CORE set with at least $2t + 1$ parties at the conclusion of Step 4, which includes at least $t + 1$ honest parties. By construction of the CORE set, it is guaranteed that the $f_i(x)$ polynomial of every honest $P_i \in \text{CORE}$ is consistent with $g_j(y)$ of every honest P_j . Suppose for the sake of contradiction that there exists some honest $P_i \in \text{CORE}$ and an honest P_j such that $f_i(j) \neq g_j(i)$. We have the following two cases:

1. If $P_j \in \text{pubR}$, the dealer must have broadcasted $g_j^D(y)$ in Step 3. If indeed $f_i(j) \neq g_j^D(i)$, then P_i would not have broadcasted OK in Step 4, which is a contradiction.
2. If $P_j \notin \text{pubR}$, and indeed $f_i(j) \neq g_j(i)$, then honest P_i, P_j would have broadcasted a mutual complaint which the dealer would have to resolve by broadcasting either $g_i^D(y)$ or $g_j^D(y)$, which is a contradiction.

Therefore, by Claim 3.3.2, there exists a unique bivariate polynomial $S(x, y)$ such that every honest $P_i \in \text{CORE} \setminus \mathcal{C}$ holds $S(x, i)$ and $S(i, y)$ and every honest P_j holds $S(j, y)$ by the conclusion of Step 4. We claim that all the honest parties output shares on this polynomial at the termination of the protocol. In particular, we prove the following.

Lemma 3.4.4. *If there exists a set H of at least $t + 1$ honest parties such that every $P_h \in H$ holding $(f_h(x), g_h(y))$ has $\text{happy}_h = 1$ and broadcasts OK Step 6, then every honest party P_i outputs g and f polynomial consistent with the unique bivariate polynomial $S(x, y)$ (see Claim 3.3.1) defined by parties in H .*

Note that, in Step 5, the dealer must have broadcasted $f_j^D(x)$ for every $P_j \notin \text{CORE}$. We now show that the dealer must broadcast $f_j(x) = S(x, j)$. Assume for the sake of contradiction that $f_j(x) \neq S(x, j)$. Since both $f_j(x)$ and $S(x, j)$ are degree- $2t$ polynomials, they can agree on at most $2t$ points. However, the number of honest parties is at least $2t + 1$. Therefore, there must exist some honest party P_l for which $f_j(l) \neq g_l(j) = S(l, j)$. We thus have the following two cases to consider:

1. If $P_l \in \text{pubR}$, then $g_l^D(y)$ was already revealed publicly in Step 3 and hence it must hold that $g_l^D(y) = S(l, y)$. This implies that parties can publicly verify the consistency of $f_j(x)$ and $g_l^D(y)$. If indeed $f_j(l) \neq g_l^D(j)$, every party in H would set its happy to 0 and not broadcast OK. Hence, this case is impossible.
2. If $P_l \notin \text{pubR}$, then $g_l(y)$ is private. This implies that P_l would not broadcast OK during Step 5, and thus the dealer must reveal $g_l^D(y)$ in Step 6. We can have two sub-cases:
 - (a) The dealer reveals $g_l^D(y) \neq S(l, y)$. Both, $g_l^D(y)$ and $S(l, y)$ are degree- t polynomials, and hence they can agree on at most t points. Moreover, since $|H| \geq t + 1$,

there must exist a party $P_h \in H$ such that $g_h(h) \neq f_h(l)$. This implies that P_h would set $\text{happy}_h = 0$ and not broadcast OK in Step 6, which is a contradiction.

- (b) The dealer reveals $g_l^D(y) = S(l, y)$. This implies that parties can publicly verify the consistency of $f_j(x)$ and $g_l^D(y)$. If indeed $f_j(l) \neq g_l^D(j)$, every party in H would set its happy to 0 and not broadcast OK. Hence, this case is impossible.

We therefore conclude that the dealer must reveal $f_j(x) = S(x, j)$ for every $P_j \notin \text{CORE}$. Thus, if indeed $2t + 1$ parties broadcast OK in Step 6, it holds that each P_i holds $f_i(x) = S(x, i)$ and $g_i(y) = S(i, y)$ on a unique $(2t, t)$ -bivariate polynomial $S(x, y)$ (see Claim 3.3.2).

Since the real and simulated executions are identical, the simulator reconstructs the unique polynomial $S(x, y)$ using the shares of the simulated honest parties in H and invokes the functionality \mathcal{F}_{VSS} with the valid polynomial $S(x, y)$. The functionality in turn sends to each P_i its shares $f_i(x) = S(x, i)$ and $g_i(y) = S(i, y)$. This is the output of honest parties in the ideal execution. This is exactly the same as output of simulated honest parties which is identical to the output of honest parties in the real execution. □

Lemma 3.4.5. *Protocol Π_{pVSS} has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast for sharing $\mathcal{O}(n)$ values (i.e., $\mathcal{O}(n \log n)$ bits) simultaneously in 9 rounds.*

In this section, we give details of our gradecast, multi-moderated secret sharing and oblivious leader election protocols. We conclude with the byzantine agreement and the parallel-broadcast using the above as building blocks.

3.5 Balanced Gradecast

In a Gradecast primitive, a dealer has an input and each party outputs a value and a grade $\{0, 1, 2\}$ such that the following properties are satisfied: **(Validity)**: If the dealer is honest then all honest parties output the dealer's input and grade 2; **(Non-equivocation)**: if two honest parties each output a grade ≥ 1 then they output the same value; and lastly **(Agreement)**: if an honest party outputs grade 2 then all honest parties output the same output and with grade ≥ 1 . We model this in terms of a functionality given in Functionality 3.5.1. The case of an honest dealer captures **validity**. Case 2a and Case 2b capture the **agreement** and **non-equivocation** respectively.

Functionality 3.5.1: $\mathcal{F}_{\text{Gradecast}}$

The functionality is parameterized by the set of corrupted parties, $I \subseteq \{1, \dots, n\}$.

1. If the dealer is honest: the dealer sends m to the functionality, and all parties receive $(m, 2)$ as output.
 2. If the dealer is corrupted then it sends some message M to the functionality.
 - (a) If $M = (\text{ExistsGrade2}, m, (g_j)_{j \notin I})$ for some $m \in \{0, 1\}^*$ and each $g_j \in \{1, 2\}$, then verify that each $g_j \geq 1$ and that at least one honest party receives grade 2. Send (m, g_j) to each party P_j .
 - (b) If $M = (\text{NoGrade2}, (m_j, g_j)_{j \notin I})$ where each $m_j \in \{0, 1\}^*$ and $g_j \in \{0, 1\}$, then verify that for every $j, k \notin I$ with $g_j = g_k = 1$ it holds that $m_j = m_k$. Then, send (m_j, g_j) to each party P_j .
-

In Section 3.5.1 we first describe a protocol that is not balanced, i.e., the total communication complexity is $\mathcal{O}(n^2L)$ but in which the dealer sends $\mathcal{O}(n^2L)$ and every other party sends $\mathcal{O}(nL)$. In Section 3.5.2 we show how to make the protocol balanced, in which each party (including the dealer) sends/receives $\mathcal{O}(nL)$ bits.

3.5.1 The Gradecast Protocol

We build our construction in Protocol 3.5.2 using the idea presented in Section 3.2.3. Recall that the gradecast used inside our protocol is the naïve gradecast with complexity $\mathcal{O}(n^2L)$ bits for L -bit message, as in [66, 68]. The security of our protocol is stated in Theorem 3.5.3.

Protocol 3.5.2: $\Pi_{\text{Gradecast}}$

Input: The dealer $P \in \{P_1, \dots, P_n\}$ holds $(t + 1)^2$ field elements $(b_{i,j})_{i,j \in \{0, \dots, t\}}$ where each $b_{i,j} \in \mathbb{F}$ that it wishes to distribute. All other parties have no input.

1. **(Dealer's polynomial distribution) The dealer:**
 - (a) The dealer views its elements as a bivariate polynomial of degree at most t in both x and y , i.e., $S(x, y) = \sum_{i=0}^t \sum_{j=0}^t b_{i,j} x^i y^j$.
 - (b) The dealer sends $S(x, y)$ to all parties.
2. **(Pair-wise Information Exchange) Each party P_i :**
 - (a) Let $S_i(x, y)$ be the polynomial received from the dealer.
 - (b) P_i sends to each party P_j the four polynomials $(S_i(x, j), S_i(j, y), S_i(x, i), S_i(i, y))$.
3. **(Informing dealer about consistency) Each party P_i :**
 - (a) Initialize $\text{Agreed}_i = \emptyset$. Let $(f_i^j(x), g_i^j(y), f_j^i(x), g_j^i(y))$ be the polynomials received from party P_j . If $f_i^j(x) = S_i(x, i)$, $g_i^j(y) = S_i(i, y)$, $f_j^i = S_i(x, j)$ and $g_j^i(y) = S_i(j, y)$ then add j to Agreed_i .

- (b) Send Agreed_i to the dealer.
4. **(Quorum forming by dealer) The dealer:**
- (a) Define an undirected graph G as follows: The nodes are $\{1, \dots, n\}$ and an edge $\{i, j\} \in G$ if and only if $i \in \text{Agreed}_j$ and $j \in \text{Agreed}_i$. Use STAR algorithm (Algorithm 3.3.5) to find a set $(C, D) \in \{1, \dots, n\}^2$ where $|C| \geq t + 1$ and $|D| \geq 2t + 1$, $C \subseteq D$, such that for every $c \in C$ and $d \in D$ it holds that $c \in \text{Agreed}_d$ and $d \in \text{Agreed}_c$.
 - (b) Let E be the set of parties that agree with at least $t + 1$ parties in C . That is, initialize $E = \emptyset$ and add i to E if $|\text{Agreed}_i \cap C| \geq t + 1$.
 - (c) Let F be the set of parties that agree with at least $2t + 1$ parties in E . That is, initialize $F = \emptyset$ and add i to F if $|\text{Agreed}_i \cap E| \geq 2t + 1$.
 - (d) If $|C| \geq t + 1$ and $|D|, |E|, |F| \geq 2t + 1$, then gradecast (C, D, E, F) . Otherwise, gradecast $(\emptyset, \emptyset, \emptyset, \emptyset)$.
5. **(First reaffirmation) Each party P_i :**
- (a) Let (C_i, D_i, E_i, F_i, g) be the message that the dealer gradecasted and let g be the associated grade.
 - (b) If (1) $g = 2$; (2) $i \in C_i$; (3) $|D_i| \geq 2t + 1$; and (4) $\text{Agreed}_i \cap D_i = D_i$; then send OK_C to all parties. Otherwise, send nothing.
6. **(Second reaffirmation) Each party P_i :**
- (a) Let C'_i be the set of parties that sent OK_C in the previous round.
 - (b) If $i \in E_i$ and $|\text{Agreed}_i \cap C_i \cap C'_i| \geq t + 1$ then send OK_E to all parties.
7. **(Third reaffirmation and propagation) Each party P_i :**
- (a) Let E'_i be the set of parties that sent OK_E in the previous round.
 - (b) If $i \in F_i$ and $|\text{Agreed}_i \cap E_i \cap E'_i| \geq 2t + 1$ then send $(\text{OK}_F, S_i(x, j), S_i(j, y))$ to each party P_j .
8. **(Final propagation) Each party P_i :** Among all messages that were received in the previous round, if there exist polynomials $f'_i(x), g'_i(y)$ that were received at least $t + 1$ times, then forward those polynomials to all. Otherwise, forward \perp .
9. **(Output) Each party P_i :** Let $((f'_1(x), g'_1(y)), \dots, (f'_n(x), g'_n(y)))$ be the messages received in the previous round. If received at least $2t + 1$ polynomials that are not \perp , then use robust interpolation to obtain a polynomial $S'(x, y)$. If there is no unique reconstruction or less than $2t + 1$ polynomials received, then output $(\perp, 0)$. Otherwise, if $S'(x, y)$ is unique, then:

- (a) If (1) P_i sent OK_F in Round 7; and (2) it received $2t + 1$ messages OK_F at the end of Round 7 from parties in F_i with the same polynomials $(f'_i(x), g'_i(y))$; then output $(S', 2)$.
 - (b) Otherwise, output $(S', 1)$.
-

Theorem 3.5.3. *Let $t < n/3$. Protocol $\Pi_{\text{Gradecast}}$ (Protocol 3.5.2) securely realizes Functionality $\mathcal{F}_{\text{Gradecast}}$ (Functionality 3.5.1) in the presence of a malicious adversary controlling at most t parties. The parties send at most $\mathcal{O}(n^3 \log n)$ bits where $\mathcal{O}(n^2 \log n)$ is the number of bits of the dealer's input.*

Proof.

Efficiency. The dealer sends each message to all parties in the first step, i.e., $\mathcal{O}(n^3 \log n)$ bits. Each party then sends a pair of univariate polynomial to each other party, i.e., sends or receive $\mathcal{O}(n \log n)$ for each pair of parties, or a total of $\mathcal{O}(n^3 \log n)$. Each party sends a set of size $\mathcal{O}(n \log n)$ to the dealer, which then gradecasts, using a naïve gradecast, sets of size $\mathcal{O}(n \log n)$. This is again a total of $\mathcal{O}(n^3 \log n)$ bits. Each party then sends a pair of univariate polynomials to each other party, i.e., each party sends or receives $\mathcal{O}(n^2 \log n)$ bits, and a total of $\mathcal{O}(n^3 \log n)$ bits.

Case 1 - The dealer is honest. In this case, the honest dealer sends m to the functionality $\mathcal{F}_{\text{Gradecast}}$ (Functionality 3.5.1) and all parties receive $(m, 2)$. Moreover, the simulation is trivial. Specifically, the protocol is deterministic and excluding the dealer, parties do not have any input in the protocol. Further, the input of the dealer is known to the simulator, which it receives from the trusted party. Therefore, the simulator can just run the protocol with the exact same inputs as in the real world, and since the protocol is deterministic, the view of the adversary in the real world and the simulated execution would be identical. Thus, all that remains to be shown is that the output of honest parties in the real-world is the same as their output in the ideal world, i.e., all the honest parties output $(m, 2)$ in the protocol execution. This requirement is captured by the following claim and proven subsequently.

Claim 3.5.4 (Validity). *If the dealer is honest and starts with input polynomial $S(x, y)$, then all honest parties output the same polynomial $S(x, y)$ and grade 2.*

Proof. If the dealer is honest, then it sends the same polynomial $S(x, y)$ to all other parties in Round 1b. This guarantees that all the honest parties are included in the Agreed set of every

honest party during Round 3a. Following this, in Round 4, the dealer finds sets (C, D, E, F) as described, and gradecasts them. By the properties of gradecast, it is guaranteed that all the honest parties receive the same sets with grade 2. Moreover, since there exists a clique of size $2t + 1$ (consisting of all the honest parties), the STAR algorithm (Algorithm 3.3.5) finds sets (C, D) as described, such that C contains at least $t + 1$ honest parties (see Claim 3.3.6). This implies that E as well as F contain all honest parties. Then:

1. First reaffirmation (Round 5) – all honest parties in C send OK_C to all parties. This is because, by the definition of the set D , all conditions in Round 5 are satisfied for each honest party P_j for $j \in C$.
2. Second reaffirmation (Round 6) – since C contains at least $t + 1$ honest parties, each honest party in E sends OK_E . Moreover, as mentioned above, all honest parties are a part of E and consequently send OK_E .
3. Third reaffirmation ((Round 7)) – as mentioned, all honest parties belong to F and also agree among themselves. Therefore, each honest party P_j sends to every party P_k , the message $(\text{OK}_F, S_j(x, k), S_j(k, y))$.

Since, for every honest P_j , $S_j(x, k)$ is equal to $S(x, k)$ (and $S_j(k, y)$ is equal to $S(k, y)$ respectively) that the dealer sent, it implies that each honest party P_k receives the same polynomial at least $2t + 1$ times in Round 8, which it forwards to all other parties. Therefore all honest parties receive at least $2t + 1$ pairs of polynomials on $S(x, y)$ thus ensuring robust reconstruction. Moreover, since each honest party P_j had sent OK_F in Round 7, and thus received $2t + 1$ OK_F messages at the end of Round 7 with the same polynomials $S(x, j), S(j, y)$, it satisfies the conditions described in step 9a and therefore outputs the polynomial that the dealer had sent, with grade 2. \square

Case 2 - The dealer is corrupt. Since the honest parties have no input, simulation is trivial. In particular, the simulator just runs the protocol while simulating the honest parties interacting with the adversary. Since the protocol is deterministic, the view produced by the simulator is exactly the same as the view of the adversary in the real world. At the end of the execution, the simulator holds the outputs of the simulated honest parties, i.e., (m_j, g_j) for every $j \notin I$ where I is the set of corrupted parties. Then:

1. If there exists an honest party P_j with grade $g_j = 2$ then the simulator verifies that every other honest party P_k holds the same message $m_k = m_j$ with grade $g_k \geq 1$. If this condition holds, then it sends $(\text{ExistsGrade2}, m_j, \{g_j\}_{j \notin I})$ to the trusted party. Otherwise, it fails and outputs \perp .

2. If all honest parties have $g_j \leq 1$, then the simulator verifies that for every $j, k \notin I$ with $g_j = g_k = 1$ it holds that $m_j = m_k$. If so, it sends $(\text{NoGrade2}, (m_j, g_j)_{j \notin I})$ to the trusted party and halts. Otherwise, it fails and outputs \perp .

The functionality then delivers the output to all the honest parties. The following claims show that the output of honest parties in the real world is the same as the output in the ideal world, and that the simulator never fails and outputs \perp .

Claim 3.5.5 (Agreement). *If the dealer is corrupted and some honest party outputs $(S', 2)$, then all honest parties output S' with grade ≥ 1 .*

Proof. If some honest party P_i outputs S' with grade 2, then as per the conditions of step 9a, it must have sent OK_F in Round 7. Moreover, the party must have unique interpolation; and received $2t + 1$ OK_F messages from parties in F_i with the same pair of polynomials $f_i(x), g_i(y)$ during Round 7.

Since $2t + 1$ parties sent OK_F messages with the same $f_i(x), g_i(y)$, it implies that there are at least $t + 1$ honest parties in F_i that sent $(\text{OK}_F, f_i(x), g_i(y))$ to P_i . For each such honest party P_j that sent this message, as per the conditions of Round 7, it holds that $j \in F_j$ and that $|\text{Agreed}_j \cap E_j \cap E'_j| \geq 2t + 1$. Here, E'_j is the set of parties that sent OK_E to party P_j at the end of Round 6. This further implies that at least $t + 1$ honest parties in E'_j sent OK_E , and moreover, the same set of honest parties sent OK_E to all the honest parties. Denote this set of honest parties as E_H .

Since there are at least $t + 1$ honest parties in E_H , for each such honest party P_j it holds that $|\text{Agreed}_j \cap C_j \cap C'_j| \geq t + 1$, where C'_j is the set of parties that sent OK_C to P_j at the end of Round 5. This in turn implies that at least one honest party belongs to C_j and has sent OK_C , and moreover, the same set of honest parties has sent OK_C to all the other honest parties. Since at least one honest party sent OK_C , due to the conditions of Round 5, it also implies that this honest party received (C, D, E, F) with grade 2 from the dealer. This means that the tuple (C, D, E, F) has been received *identically* by all the honest parties (with grade ≥ 1) and hence we can refer to each of the sets C, D, E and F without party-specific subscript. With the above observations, we now prove the aforementioned claim using the following three claims which are proved subsequently.

Claim 3.5.6. *All honest parties in C that sent OK_C must hold the same bivariate polynomial, denoted as $\hat{S}(x, y)$.*

Claim 3.5.7. *If an honest party P_k such that $k \in E$ sent OK_E in Round 6, then there is at least one honest party that sent OK_C in Round 5 and it holds that $S_k(x, k) = \hat{S}(x, k)$ and $S_k(k, y) = \hat{S}(k, y)$, where \hat{S} is defined by the honest parties in C .*

Claim 3.5.8. *If an honest party P_j such that $j \in F$ sent OK_F in Round 7, then there are at least $t + 1$ honest parties that sent OK_E in Round 6, and at least one honest party that sent OK_C in Round 5. Moreover, it holds that $S_j(x, y) = \hat{S}(x, y)$, where \hat{S} is defined by the honest parties in C .*

Given these three claims, together with the fact that $2t + 1$ OK_F messages were received by the honest party that output grade 2 at the end of the protocol, we have that $t + 1$ honest parties hold the same polynomial $\hat{S}(x, y)$ in Round 7. Each such honest party sends to each party P_j the two polynomials $\hat{S}(x, j), \hat{S}(j, y)$ in Round 7. Thus, each honest party receives the same two polynomials at least $t + 1$ times and forwards them to every other party. Note that no other polynomial can be received with plurality $t + 1$. This implies that every honest party receives $2t + 1$ pairs of polynomials on the polynomial \hat{S} , thus ensuring a robust interpolation. Therefore, every honest party outputs \hat{S} with grade at least 1.

Proof of Claim 3.5.6: Let C_H be the set of honest parties in C that sent OK_C . Consider $j, k \in C_H$. We show that $S_j(x, y) = S_k(x, y)$, where S_j, S_k are the polynomials received by parties P_j, P_k respectively in Step 1b. As per the conditions of Round 5, an honest party P_j sends OK_C only if it receives (C_j, D_j, E_j, F_j) with grade 2, $j \in C_j$, $|D_j| \geq 2t + 1$ and $\text{Agreed}_j \cap D_j = D_j$. Since the message is received with grade 2 by some honest P_j , by the properties of gradecast we know that all honest parties receive the same message (C_j, D_j, E_j, F_j) and therefore we can omit the subscript of D_j . Moreover, since $|D| \geq 2t + 1$, it holds that D contains at least $t + 1$ honest parties. Further, owing to the fact that $\text{Agreed}_j \cap D = D$ and $\text{Agreed}_k \cap D = D$, we have $S_j(x, d) = S_k(x, d)$ and $S_j(d, y) = S_k(d, y)$ for every honest d in D . Since D has at least $t + 1$ honest parties, it holds that $S_j(x, y) = S_k(x, y)$. \square

Proof of Claim 3.5.7: Let E_H denote the set of honest parties that sent OK_E in Round 6. For each honest party P_k in E_H , it holds that $|\text{Agreed}_k \cap C_k \cap C'_k| \geq t + 1$ where C'_k is the set of parties that sent OK_C in Round 5. Let C_H be the set of honest parties that sent OK_C in Round 5. It must hold that $|C_H| \geq 1$, and therefore at least one honest party sent OK_C to all honest parties. Therefore, by the condition of Round 5 and the properties of gradecast, all the honest parties receive the same sets (C, E, D, F) and we thus can omit the subscript of C_k . Moreover, from Claim 3.5.6, we have that all honest parties in C_H hold the same bivariate polynomial, $\hat{S}(x, y)$.

Further, since $|\text{Agreed}_k \cap C \cap C'_k| \geq t + 1$ for each $k \in E_H$ we have that P_k agrees with at least one honest party in C_H . Let $j \in C_H$ such that $j \in \text{Agreed}_k$. In Step 2b, P_j and P_k exchanged polynomials and since $j \in \text{Agreed}_k$, it must hold that $k \in \text{Agreed}_j$. This implies that P_k holds the two polynomials $\hat{S}(x, k), \hat{S}(k, y)$ in round 6. \square

Proof of Claim 3.5.8: For each party P_j such that $j \in F$ that sends OK_F at the end of Round 7, it holds that $|\text{Agreed}_j \cap E_j \cap E'_j| \geq 2t + 1$ where E'_j is the set of parties that sent OK_E to P_j in Round 6. This implies that at least $t + 1$ honest parties sent OK_E , which further implies that at least one honest party sent OK_C at the end of Round 5 (see claim 3.5.7). Moreover, as described in the proof of claims 3.5.6 and 3.5.7, due to the conditions of Round 5 and properties of gradecast, this also implies that we can omit the subscript of E_j since all honest parties see the same sets (C, D, E, F) . Let E_H be the set of honest parties that sent OK_E in Round 6, where we have that $|E_H| \geq t + 1$.

From Claim 3.5.7 we know that each party P_k in E_H holds $\hat{S}(x, k), \hat{S}(k, y)$. For each honest party P_j such that $j \in F$ that sent OK_F we have that P_j agrees with $t + 1$ parties in E_H . For each such $k \in E_H$, we conclude that $S_j(x, k) = \hat{S}(x, k)$ and $S_j(k, y) = \hat{S}(k, y)$. Since this holds for at least $t + 1$ distinct indices in E_H , we conclude that $S_j(x, y) = \hat{S}(x, y)$. \square

This concludes the proof of Claim 3.5.5. \square

Claim 3.5.9 (Non-equivocation). *If two honest parties each output a grade ≥ 1 then they output the same polynomial $\hat{S}(x, y)$.*

Proof. If an honest party outputs grade ≥ 1 then it must have received at least $2t + 1$ pairs of polynomials at the end of Round 8. This implies that at least $t + 1$ honest parties forwarded their polynomials in Round 8 thus indicating that each of these honest parties received their polynomials with plurality at least $t + 1$ at the end of Round 7. This in turn implies that there is at least one honest party that sent OK_F in Round 7. By virtue of claim 3.5.8, we know that all honest parties that sent OK_F in Round 7 hold the same polynomial $\hat{S}(x, y)$. As a result, the only two polynomials that can be forwarded by honest parties in Round 8 (i.e., that can be received $t + 1$ times) must lie on $\hat{S}(x, y)$. Since $t + 1$ honest parties forwarded their polynomials, it holds that all honest parties receive at least $t + 1$ univariate shares on $\hat{S}(x, y)$.

Thus, given that any two degree- t bivariate polynomials can agree on at most t shares, for any honest party that receives $2t + 1$ pairs of polynomials and runs the robust interpolation, the only polynomial that can be accepted is $\hat{S}(x, y)$. \square

This concludes the proof of Theorem 3.5.3. \square

3.5.2 Making the Protocol Balanced

To make the protocol balanced, note that each party sends or receives $\mathcal{O}(n^2 \log n)$ bits except for the dealer who sends $\mathcal{O}(n^3 \log n)$. We therefore change the first round of the protocol as follows:

1. The dealer:

- (a) The dealer views its elements as a bivariate polynomial of degree at most t in both x and y , i.e., $S(x, y) = \sum_{i=0}^t \sum_{j=0}^t b_{i,j} x^i y^j$.
- (b) The dealer sends $S(x, i)$ to each party P_i .

2. Each party P_i :

- (a) Forwards the message received from the dealer to every other party.
- (b) Given all univariate polynomials received, say $u(x, 1), \dots, u(x, n)$, runs the Reed-Solomon decoding procedure to obtain the bivariate polynomial $S_i(x, y)$. If there is no unique decoding, then use $S_i(x, y) = \perp$.

- 3. Continue to run Protocol $\Pi_{\text{Gradecast}}$ (Protocol 3.5.2) from Step 2 to the end while interpreting $S_i(x, y)$ decoded from the prior round as the polynomial received from the dealer.

Theorem 3.5.10. *The modified protocol securely realizes Functionality $\mathcal{F}_{\text{Gradecast}}$ (Functionality 3.5.1) in the presence of a malicious adversary controlling at most t parties. Each party, including the dealer sends or receives $\mathcal{O}(n^2 \log n)$ bits (giving a total communication complexity of $\mathcal{O}(n^3 \log n)$).*

Proof. We show that the above procedure is equivalent to let the dealer send S to each party separately.

The case of an honest dealer. In that case, the dealer holds a (t, t) -bivariate polynomial $S(x, y)$ and sends to each party its share on S , which is essentially a univariate polynomial with degree- t in x . Since there are at least $2t + 1$ honest parties, each honest party holds at least $2t + 1$ univariate shares (in x) on the polynomial S defined by the dealer and therefore the decoding will result in the polynomial S that the dealer initially holds.

The case of a corrupted dealer. In this case, any attack by the adversary in the modified steps of the protocol can be mapped to a malicious behaviour of the dealer in the polynomial distribution step of the unbalanced protocol. Specifically, for an honest party P_j , the failure to obtain a bivariate polynomial $S_j(x, y)$ upon decoding in the balanced protocol is equivalent to a corrupt dealer sending \perp to P_j in the first step of the unbalanced protocol. Thus, for any

adversary \mathcal{A} attacking the modified protocol we can build an adversary \mathcal{A}' for $\Pi_{\text{Gradecast}}$. Since $\Pi_{\text{Gradecast}}$ tolerates at most t parties and securely realizes Functionality $\mathcal{F}_{\text{Gradecast}}$, we get that the modified protocol also securely realizes Functionality $\mathcal{F}_{\text{Gradecast}}$. \mathcal{A}' corrupts the same set of parties as \mathcal{A} . It simulates the honest parties in the first two rounds of the modified protocol and obtains the polynomial S_j that each honest party P_j uses. It then simply hands S_j as the polynomial the corrupted dealer sends to P_j in Protocol $\Pi_{\text{Gradecast}}$. \square

3.5.3 Conclusions

The following is a simple corollary, where for general message length of L bits the dealer simply breaks the message into $\ell = \lceil L/(t+1)^2 \log n \rceil$ blocks and runs ℓ parallel executions of gradecast. Each party outputs the concatenation of all executions, with the minimum grade obtained on all executions. The protocol is optimal for $L > n^2 \log n$. We thus obtain the following corollary.

Corollary 3.5.11. *Let $t < n/3$. There exists a gradecast protocol in the presence of a malicious adversary controlling at most t parties, where for transmitting L bits, the protocol requires the transmission of $\mathcal{O}(nL + n^3 \log n)$ bits, where each party sends or receives $\mathcal{O}(L + n^2 \log n)$ bits.*

3.6 Multi-Moderated Packed Secret Sharing

At a high level multi-moderated packed secret sharing is a packed VSS moderated by a set \mathcal{M} of $t+1$ distinguished parties called moderators. The parties output a flag for every moderator in the end. We represent the flag for a moderator $M \in \mathcal{M}$ held by a party P_k as v_M^k . In addition, each party P_k holds a variable d_M^k taking values from $\{\text{accept}, \text{reject}\}$ for each $M \in \mathcal{M}$ which identifies whether the dealer is accepted or rejected when M assumes the role of the moderator.

If a moderator M is honest, then every honest party P_k will set $v_M^k = 1$ and the properties of VSS will be satisfied irrespective of whether the dealer is honest or corrupt. If the dealer is honest, every honest P_k will set $d_M^k = \text{accept}$. For a corrupt dealer, the bit can be 0 or 1 based on the dealer's behaviour, but all the honest parties will unanimously output the same outcome.

If a moderator M is corrupt, then it is guaranteed that: if some honest party P_k sets the flag $v_M^k = 1$, then the properties of VSS will be satisfied irrespective of whether the dealer is honest or corrupt. That is, if the dealer is honest every honest P_k outputs $d_M^k = \text{accept}$. For a corrupt dealer, it is guaranteed that all the honest parties unanimously output the same outcome for the dealer. We note that when no honest party sets its flag to 1 for a moderator

M , then irrespective for whether the dealer is honest or corrupt, it is possible that the parties do not have agreement on their d_M^k .

We capture these properties in a functionality, defined as follows:

Functionality 3.6.1: $\mathcal{F}_{\text{mm-pVSS}}$ – **Multi-Moderated Packed Secret Sharing**

The functionality is parameterized by the set of corrupted parties $I \subseteq \{1, \dots, n\}$, a set \mathcal{M} of $t + 1$ distinguished parties called as moderators.

1. The dealer sends polynomials $f_j(x), g_j(y)$ for every j . If the dealer is honest, then there exists a single $(2t, t)$ polynomial $S(x, y)$ that satisfies $f_j(x) = S(x, j)$ and $g_j(y) = S(j, y)$ for every $j \in \{1, \dots, n\}$.
 2. If the dealer is honest, then send $f_i(x), g_i(y)$ for every $i \in I$ to the adversary.
 3. For each moderator $M_j \in \mathcal{M}$:
 - (a) If the moderator M_j is honest, then set $v_{M_j}^k = 1$ for every $k \in \{1, \dots, n\}$. Moreover:
 - i. If the dealer is honest, then set $d_{M_j}^k = \text{accept}$ for every $k \in \{1, \dots, n\}$.
 - ii. If the dealer is corrupt, then receive a message m_j from the adversary. If $m_j = \text{accept}$ then verify that the shares of the honest parties define a unique $(2t, t)$ -polynomial. If so, set $d_{M_j}^k = \text{accept}$ for every $k \in \{1, \dots, n\}$. In any other case, set $d_{M_j}^k = \text{reject}$ for every $k \in \{1, \dots, n\}$.
 - (b) If the moderator M_j is corrupt then receive m_j from the adversary.
 - i. If $m_j = (\text{Agreement}, (v_{M_j}^k)_{k \notin I}, d_{M_j})$ where $d_{M_j} \in \{\text{accept}, \text{reject}\}$, and for some $k \notin I$ it holds that $v_{M_j}^k = 1$. Set $(v_{M_j}^k)_{k \notin I}$ as received from the adversary. Verify that $S(x, y)$ is $(2t, t)$ -polynomial. If not, set $d_{M_j}^k = \text{reject}$ for every $k \notin I$. Otherwise, set $d_{M_j}^k = d_{M_j}$ for every $k \notin I$.
 - ii. If $m_j = (\text{NoAgreement}, (d_{M_j}^k)_{k \notin I})$ where each $d_{M_j}^k \in \{\text{accept}, \text{reject}\}$, then set $v_{M_j}^k = 0$ for every $k \in \{1, \dots, n\}$ and $d_{M_j}^1, \dots, d_{M_j}^n$ as received from the adversary.
 4. **Output:** Each honest party P_k ($k \notin I$) receives as output $f_i(x), g_i(y), (d_M^k)_{M \in \mathcal{M}}$, and flags $(v_M^k)_{M \in \mathcal{M}}$.
-

To clarify, each party P_i receives global shares for all moderators, and an output d_M^i and flag v_M^i per each moderator $M \in \mathcal{M}$. If the dealer and the moderator are honest, then all the flags are 1 and the parties accept the shares. If the moderator M_j is corrupted, then as long as there is one honest party P_k with $v_{M_j}^k = 1$ there will be an agreement in the outputs $d_{M_j}^1, \dots, d_{M_j}^n$ (either all the honest parties accept or all of them reject). When $v_{M_j}^k = 0$ for all the honest parties, we might have inconsistency in the outputs $d_{M_j}^1, \dots, d_{M_j}^n$ with respect to that moderator.

The protocol. We build on the discussion given in Section 3.2.1. We consider the protocol of VSS where the dealer inputs some bivariate polynomial $S(x, y)$ of degree at most $2t$ in x and degree at most t in y . For multi-moderated packed secret sharing, essentially, each broadcast from Π_{pVSS} is simulated with two sequential gradecasts. The first gradecast is performed by the party which intends to broadcast in the underlying packed VSS protocol, while the second is executed by a moderator. Note that these gradecasts are realized via the protocol $\Pi_{\text{Gradecast}}$, presented in the Section 3.5, having the optimal communication complexity. Up to Step 6 of Π_{pVSS} (Protocol 3.4.2), the dealer is the moderator for each gradecast. At Step 6, we fork into $t + 1$ executions, with a unique party acting as the moderator in each execution. Since the protocol steps remain similar to Π_{pVSS} , we describe the multi-moderated packed secret sharing protocol below in terms of how the broadcast is simulated at each step and the required changes at Step 6 of the packed VSS protocol.

Protocol 3.6.2: $\Pi_{\text{mm-pVSS}}$ – Multi-Moderated Packed Secret Sharing

Simulating broadcast up to (including) Step 6 of Π_{pVSS} :

1. Simulating broadcast of a message by the dealer.
 - (a) **The dealer:** When the dealer has to broadcast a message m it gradecasts it.
 - (b) **Party P_i :** Let (m, g) be the message gradecasted by the dealer, where m is the message and g is the grade. Proceed with m as the message broadcasted by the dealer. If $g \neq 2$, then set $\text{happy}_i = 0$ within the execution of Π_{pVSS} .
2. Simulating broadcast of a party P_j .
 - (a) **Party P_j :** When P_j wishes to broadcast a message m , it first gradecasts it.
 - (b) **The dealer:** Let (m, g) be the message and g its associated grade. The dealer gradecasts m .
 - (c) **Each party P_i :** Let (m', g') be the messages gradecasted by the dealer. Use m' as the message broadcasted by P_j in the protocol. Moreover, if $g' \neq 2$; or if $g = 2$ but $m' \neq m$, then P_i sets $\text{happy}_i = 0$ within the execution of Π_{pVSS} .

After Step 6 of Π_{pVSS} :

1. **Each party P_i :** Set $v_{M_j}^i = 1$, and let $f_i(x), g_i(y)$ be the pair of shares P_i is holding at end of Step 6. Gradecast accept if $\text{happy}_i = 1$ and reject otherwise.
At this point, we fork into $|\mathcal{M}|$ executions, one per moderator $M_j \in \mathcal{M}$ as follows:
 - (a) **The moderator M_j :** Let (a_1, \dots, a_n) be the decisions of all parties as received from the gradecast. Gradecast (a_1, \dots, a_n) .

- (b) **Each party P_i :** Let (a_1, \dots, a_n) be the decisions received directly from the parties, and let (a'_1, \dots, a'_n) be the message gradecasted from the moderator M_j with associated grade g' . Set $v_{M_j}^i = 0$ if $g' \neq 2$, or there exists a_k received from P_k with grade 2 but for which $a_k \neq a'_k$. Then:
- i. If there exists $2t + 1$ accepts within (a'_1, \dots, a'_n) , then set $d_{M_j}^i = \text{accept}$.
 - ii. Otherwise, set $d_{M_j}^i = \text{reject}$.

2. **Output:** P_i outputs $(f_i(x), g_i(y))$, $(d_{M_1}^i, \dots, d_{M_t}^i)$ and $(v_{M_1}^i, \dots, v_{M_t}^i)$.

Theorem 3.6.3. *Let $t < n/3$. Protocol $\Pi_{\text{mm-pVSS}}$ (Protocol 3.6.2) computes $\mathcal{F}_{\text{mm-pVSS}}$ (Functionality 3.6.1) in the presence of a malicious adversary corrupting at most t parties. The protocol requires the transmission of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels, the dealer gradecasts $\mathcal{O}(n^2 \log n)$ bits, and each party gradecasts at most $n \log n$ bits.*

Proof. Case 1 - The dealer is honest. Let \mathcal{S}_{VSS} be the simulator of the VSS protocol (Protocol 3.4.2). The simulator receives from the trusted party, the shares of the corrupted parties – $S(x, i), S(i, y)$ as in Step 2 of the functionality, and simulates the view of the adversary similar to \mathcal{S}_{VSS} . While all the modifications described in the protocol for $\Pi_{\text{mm-pVSS}}$ (Protocol 3.6.2) are deterministic, the simulator now also has to emulate Functionality 3.5.1. We claim that when the dealer is honest, no honest party sets $\text{happy}_i = 0$. This holds from the same reasoning as in \mathcal{S}_{VSS} , and due to the fact that the modifications in the multi-moderated packed secret sharing do not affect this claim. In particular, note that the dealer's broadcasted messages, which are now emulated using gradecast will always be received with grade 2 by all the honest parties. Furthermore, observe that every other party's broadcast which is emulated via that party's gradecast followed by dealer's gradecast will not lead to an honest party setting $\text{happy}_i = 0$. This is because neither an honest dealer's gradecast will have grade less than 2, nor a gradecast by some party which is received by an honest party with grade 2 will follow the dealer's gradecast with a different output value. The latter holds due to the property of gradecast, where if an honest party receives a message with grade 2 then all the honest parties (including the dealer) would receive the same message with grade ≥ 1 , and the dealer would further gradecast the same message. As a result, no honest party sets $\text{happy}_i = 0$.

Consequently, at the modified Step 6, an honest party's initial decision will be accept, and the final decision of whether to accept or reject, and the grade of the moderator is determined by the messages it receives from the moderator and the decisions it received from other parties. Thus, the simulator can completely simulate this stage as well for all moderators

$M_j \in \mathcal{M}$. It can then obtain the outputs of all honest parties, i.e., the set $(d_{M_j}^k, v_{M_j}^k)_{k \notin I, M_j \in \mathcal{M}}$, although it cannot necessarily compute the output share of each honest party. For each moderator $M_j \in \mathcal{M}$:

1. If the moderator is honest, then the trusted party does not expect a message from the simulator and just delivers to each honest party $d_{M_j}^k = \text{accept}$ and $v_{M_j}^k = 1$. We will show below (Claim 3.6.4) that the same happens in the simulated execution, and in the real execution.
2. If there exists a simulated honest party that outputs $v_{M_j}^k = 1$, then we prove below (Claim 3.6.5) that for every honest P'_k we have that $d_{M_j}^{k'} = d_{M_j}^k$. Then, the simulator sends $m_j = (\text{Agreement}, (v_{M_j}^k)_{k \notin I})$ to the trusted party. Since the dealer is honest, $S(x, y)$ is of degree $(2t, t)$, and the trusted party would just forward $d_{M_j}^k, v_{M_j}^k$ to every honest party P_k .
3. If for every $k \notin I$ it holds that $v_{M_j}^k = 0$ then the simulator sends $m_j = (\text{NoAgreement}, (d_{M_j}^k)_{k \notin I})$ to the trusted party. The trusted party just forwards $d_{M_j}^k$ to every honest party P_k with grade 0 (as in the simulated execution).

The above shows that the output of all honest parties in the ideal world is exactly the same as the output of honest parties simulated by \mathcal{S}_{VSS} in the simulated execution. As mentioned in the proof of Π_{pVSS} (Theorem 3.4.3), the view of the adversary in the simulation of \mathcal{S}_{VSS} and in the real execution is exactly the same. We now show that the output of all honest parties in the real execution is exactly the same as the output of all simulated honest parties in the simulated execution. To conclude the case of an honest dealer, we have the following claims:

Claim 3.6.4. *When the dealer is honest and the moderator M_j is honest, every honest party P_k sets $d_{M_j}^k = \text{accept}$ and $v_{M_j}^k = 1$.*

Proof. Note that at Step 6, if some honest party receives the decision a_i with grade 2 from a party P_i , then by the properties of gradecast, the honest moderator M_j receives the same a_i (with grade ≥ 1) and further gradecasts the exact same message. Moreover, the message (a_1, \dots, a_n) gradecasted by an honest moderator is received by all the honest parties with grade 2. Thus, for every honest party P_k it holds that $v_{M_j}^k = 1$. Additionally, as described, for the case of an honest dealer, for every honest party P_i it holds that $\text{happy}_i = 1$ and hence every honest party gradecasts its initial decision as accept , ensuring a total of at least $2t + 1$ accept decisions. Thus, for every honest party P_k , it holds that $d_{M_j}^k = \text{accept}$. Note that in the ideal world, this corresponds to Step 3(b)i of the functionality (Functionality 3.6.1), where similar to the real world, all honest parties output $d_{M_j}^k = \text{accept}$ and $v_{M_j}^k = 1$.

□

Claim 3.6.5. *If the dealer is honest and the moderator M_j is corrupted then if some honest party P_k holds $v_{M_j}^k = 1$ then all other honest parties P'_k have $d_{M_j}^{k'} = \text{accept}$.*

Proof. If an honest party P_k outputs $v_{M_j}^k = 1$, then it must have received the decisions (a'_1, \dots, a'_n) from M_j with grade 2, and moreover, all the decisions corresponding to honest parties must agree with those gradecasted previously by the respective honest parties themselves. As described earlier, for the case of an honest dealer, every honest party gradecasts its initial decision as accept. This implies that, of (a'_1, \dots, a'_n) , at least $2t + 1$ decisions are accept. Furthermore, as guaranteed by gradecast, all the other honest parties see the exact same decisions (a'_1, \dots, a'_n) for the moderator M_j , though they might have grade equal to 1. As a result, every other honest party P'_k sets $d_{M_j}^{k'} = \text{accept}$ while $v_{M_j}^{k'}$ is either 0 or 1 (depending on the grade of the message (a'_1, \dots, a'_n) gradecasted by M_j). In the ideal execution, the simulator in this case sends $(\text{Agreement}, \cdot)$ as described above and the outputs $d_{M_j}^k$ are the same for all honest P_k . □

Case 2 - The dealer is corrupt. When the dealer is corrupted, simulation is trivial since all honest parties have no input. The simulator just simulates the honest parties interacting with the adversary as in the protocol specifications, and simulates the behavior of the gradecast functionality as in Functionality 3.5.1. Moreover, since the protocol is deterministic, the view of the adversary in the real execution and in the ideal execution is exactly the same.

The simulator proceeds by simulating all the honest parties' messages and let $(f_i(x), g_i(y), (d_{M_1}^i, \dots, d_{M_t}^i), (v_{M_1}^i, \dots, v_{M_t}^i))$ be the output of the simulated honest party P_i . The decision gradecasted by an honest party is the same for all the moderators. The proof of packed VSS (Theorem 3.4.3) shows that if there are at least $t + 1$ honest parties that decide on accept in Step 6, then there exists a unique bivariate polynomial $S(x, y)$ of degree- $(2t, t)$ such that the shares of all the honest parties lie on this polynomial. Then, for each moderator M_j :

1. **If M_j is honest:** We will show below (Claim 3.6.6) that the output of every simulated honest party P_k is $v_{M_j}^k = 1$ and that all the honest parties have the same output $d_{M_j}^k$. If all of them have $d_{M_j}^k = \text{accept}$, then the simulator sets $m_j = \text{accept}$. If all have $d_{M_j}^k = \text{reject}$, then it sets $m_j = \text{reject}$.
2. **If M_j is corrupted:** We have the following sub-cases to consider.

- (a) **There exists an honest party P_k with $v_{M_j}^k = 1$ and $d_{M_j}^k = \text{accept}$:** We show below (Claim 3.6.7) that every other honest party P'_k must have $d_{M_j}^{k'} = \text{accept}$, and the shares of all the honest parties lie on a unique bivariate polynomial $S(x, y)$ of degree- $(2t, t)$. In this case, the simulator sets $m_j = (\text{Agreement}, (v_{M_j}^k)_{k \notin I}, \text{accept})$.
- (b) **There exists an honest party P_k with $v_{M_j}^k = 1$ and $d_{M_j}^k = \text{reject}$:** We show subsequently (Claim 3.6.7) that every other honest party P'_k must have $d_{M_j}^{k'} = \text{reject}$. Thus the simulator sets $m_j = (\text{Agreement}, (v_{M_j}^k)_{k \notin I}, \text{reject})$.
- (c) **All honest parties have $v_{M_j}^k = 0$:** In this case, there is no guarantee on the output. Therefore, the simulator sets $m_j = (\text{NoAgreement}, (d_{M_j}^k)_{k \notin I})$.

The simulator sends m_j for every $M_j \in \mathcal{M}$ to the trusted party. It is easy to see by inspection of the functionality (Functionality 3.6.1) that upon sending these messages to the trusted party, all the honest parties in the ideal world receive the exact same output as the simulated honest parties in the simulated execution. Further, since the simulated execution is exactly the same as the corresponding execution in the real world, the outputs of honest parties and views are identical. To conclude the proof, we prove the following claims.

Claim 3.6.6. *For an honest moderator M_j , each honest party P_k outputs a grade $v_{M_j}^k = 1$. Moreover, all the honest parties have the same output $d_{M_j}^k$, i.e., either all accept or all reject.*

Proof. Note that all the honest parties receive the messages gradecasted by the honest moderator M_j identically and with grade 2. Moreover, for every message gradecasted by some party which is received with grade 2 by any honest party, the properties of gradecast ensure that the moderator receives the same message (although with grade ≥ 1). This implies that the moderator gradecasts exactly the same message. In particular, this also holds true for the gradecast of the decisions (a'_1, \dots, a'_n) where each a_i was previously gradecasted by party P_i . Hence, for every honest P_k it must hold that $v_{M_j}^k = 1$. Moreover, by the property of gradecast, since all the honest parties receive the same (a'_1, \dots, a'_n) , if the decisions contain at least $2t + 1$ accept messages, then every honest party P_k sets $d_{M_j}^k = \text{accept}$. On the other hand, if the decisions do not contain $2t + 1$ accept messages, then all honest parties set $d_{M_j}^k = \text{reject}$.

For the former case, we have that at least $t + 1$ honest parties decided on accept. Therefore, according to Lemma 3.4.4, this implies that all honest parties have shares on the same $(2t, t)$ polynomial $S(x, y)$. □

Claim 3.6.7. *For a corrupted moderator M_j , if some honest party P_k holds $v_{M_j}^k = 1$ then every other honest party P'_k holds $d_{M_j}^{k'} = d_{M_j}^k$.*

Proof. If an honest party P_k outputs $v_{M_j}^k = 1$ with $\text{output}_{M_j}^k = \text{accept}$, then it must have received $2t + 1$ decisions (a'_1, \dots, a'_n) of accept from the moderator with grade 2 such that the decisions corresponding to the honest parties actually agree with the ones gradecasted by the honest parties themselves. Due to the property of gradecast, this implies that all honest parties receive the same decisions (a'_1, \dots, a'_n) . Therefore, all other honest parties P'_k also set $d_{M_j}^{k'} = \text{accept}$. We now show that there is a unique polynomial $S(x, y)$ of degree at most $2t$ in x and degree t in y , and all the honest parties output shares on this polynomial.

Since the decisions of honest parties are received with grade 2, and as described these agree with the decisions of the corresponding honest parties in (a'_1, \dots, a'_n) , there must be at least $t + 1$ honest parties that decided accept . For each such honest party P_i it holds that $\text{happy}_i = 1$. This implies that all the broadcasts within the VSS protocol were simulated by the dealer (as a moderator) correctly: P_i received all the gradecasts made by the dealer with grade 2, each complaint in the protocol that an honest party made was received with grade 2 and was echoed by the dealer with grade 2. Since there are $t + 1$ honest parties with $\text{happy}_i = 1$, we have that all the complaints of honest parties were publicly resolved by the dealer, and received by all other honest parties with grade at least 1. Thus, all honest parties see the same messages, and all parties that are not happy have public shares. By the properties of the underlying VSS protocol, all shares of honest parties lie on the same bivariate polynomial $S(x, y)$.

If an honest party P_k outputs $v_{M_j}^k = 1$ with $d_{M_j}^k = \text{reject}$, then every honest party P'_k sets $d_{M_j}^{k'} = \text{reject}$. The proof follows similar to the previous case. Specifically, if P_k outputs $v_{M_j}^k = 1$ with $d_{M_j}^k = \text{reject}$, this implies that P_k must have received (a'_1, \dots, a'_n) gradecasted by M_j with grade 2 such that the decisions of honest parties in (a'_1, \dots, a'_n) agree with the ones gradecasted by the respective parties themselves. Thus, all honest parties receive the same (a'_1, \dots, a'_n) . Moreover, (a'_1, \dots, a'_n) does not contain $2t + 1$ accept messages. Thus, each honest party P'_k sets $d_{M_j}^{k'} = \text{reject}$. \square

Efficiency. As in the packed VSS protocol, the sharing phase requires $\mathcal{O}(n^2 \log n)$ bits of point-to-point communication. Every party gradecasts $\mathcal{O}(n)$ values in the worst case to communicate its complaints, while the dealer performs $\mathcal{O}(n^2)$ gradecasts to resolve the complaints. \square

3.6.1 Reconstruction

The reconstruction protocol ensures that even for a corrupt moderator, all the honest parties reconstruct the same value when its flag is set to 1 by some honest party. This aligns with the guarantees of the sharing phase, which en-

sures that the protocol achieves VSS corresponding to a moderator when there exists an honest party with its flag set to 1 at the end of the sharing phase.

Protocol 3.6.8: $\Pi_{\text{mm-pVSS}}^{\text{Rec}}$ – **Reconstruct of Multi-Moderated Packed Secret Sharing**

The protocol is parameterized by the set of moderators \mathcal{M} and a set B containing $|\mathcal{M}|$ distinct non-zero values in the field. To be specific B denotes the set $\{-t, \dots, 0\}$ used in Π_{pVSS} . We assume a one-to-one mapping between \mathcal{M} and $\{-t, \dots, 0\}$.

Input: Each party P_i holds $(f_i(x), g_i(y)), (d_M^i)_{M \in \mathcal{M}}$ and $(v_M^i)_{M \in \mathcal{M}}$.

1. Each party sends $f_i(x)$ to all. Let $(f_1(x)', \dots, f_n(x)')$ be the polynomials received.
2. For each $M \in \mathcal{M}$ (let $\beta^* \in B$ be its associated value):
 - (a) If $d_M^i = \text{accept}$, then use Reed Solomon decoding procedure to reconstruct the unique degree- t polynomial $g_{\beta^*}(y)$ that agrees with at least $2t + 1$ values $f_1(\beta^*), \dots, f_n(\beta^*)$ and set $s_M^i = g_{\beta^*}(0)$. If there is not unique decoding, then set $s_M^i = 0$.
 - (b) If $d_M^i = \text{reject}$, then set $s_M^i = 0$.

3. **Output:** Output $(s_M^i)_{M \in \mathcal{M}}$.

Theorem 3.6.9. *For each moderator $M \in \mathcal{M}$, if there exists an honest party with $v_M^k = 1$ then all honest parties hold the same $s_M^{k'} = s_M^k$.*

Proof. By the properties of $\mathcal{F}_{\text{mm-pVSS}}$ (Functionality 3.6.1), if there exists an honest party P_k that holds $v_M^k = 1$ for some moderator M then all honest parties hold the same value d_M , i.e., all accept or reject. For each $d_M = \text{reject}$, every honest party P_i sets $s_M^i = 0$ in the reconstruction protocol. On the other hand, if there exists an honest party with $d_M = \text{accept}$, then according to Functionality 3.6.1, even in the case of a corrupted dealer we have that all the shares of honest parties lie on the same $(2t, t)$ -bivariate polynomial $S(x, y)$. As a result, each polynomial $g_{\beta^*}(y) = S(\beta^*, y)$ is of degree- t . Moreover, the parties send their shares to one another, and thus the $2t + 1$ honest parties send the degree- $2t$ polynomials $S(x, i)$ to one another. Therefore, each honest party would have at least $2t + 1$ correct points on $S(\beta^*, y)$, for which the Reed-Solomon error correction will return a unique decoding, thus ensuring that every honest party P_i obtains the same output $s_M^i = S(\beta^*, 0)$. \square

3.7 Oblivious Leader Election

We start with the functionality which captures OLE with fairness δ , where each party P_i outputs a value $\ell_i \in \{1, \dots, n\}$ such that with probability at least δ there exists a value $\ell \in \{1, \dots, n\}$ for which the following conditions hold: (a) each honest P_i outputs $\ell_i = \ell$, and (b) P_ℓ is an honest party. The functionality is parameterized by the set of corrupted parties I , a parameter $\delta > 0$ and a family of efficiently sampling distributions $\mathcal{D} = \{D\}$. Each $D \in \mathcal{D}$ is a distribution $D : \{0, 1\}^{\text{poly}(n)} \rightarrow \{1, \dots, n\}^n$ satisfying: $\Pr_{r \leftarrow \{0, 1\}^{\text{poly}(n)}} [D(r) = (j, \dots, j) \text{ s.t. } j \notin I] \geq \delta$.

Functionality 3.7.1: \mathcal{F}_{OLE} – Oblivious Leader Election Functionality

The functionality is parameterized by the set of corrupted parties $I \subset \{1, \dots, n\}$ and the family \mathcal{D} .

1. The functionality receives from the adversary a sampler D and verifies that $D \in \mathcal{D}$. If not, then it takes some default sampler in $D \in \mathcal{D}$.
 2. The functionality chooses a random $r \leftarrow \{0, 1\}^{\text{poly}(n)}$ and samples $(\ell_1, \dots, \ell_n) = D(r)$.
 3. It hands r to the adversary and it hands ℓ_i to every party P_i .
-

Looking ahead, our protocol will define a family \mathcal{D} in which the functionality can efficiently determine whether a given sampler D is a member of \mathcal{D} . Specifically, we define the sampler as a parametrized algorithm with some specific values hardwired. Therefore, the ideal adversary can just send those parameters to the functionality to specify D in the family.

Protocol 3.7.2: Π_{OLE} – Oblivious Leader Election Protocol

1. **Choose and commit weights:** Each party $P_i \in \mathcal{P}$ acts as the dealer and chooses $c_{i \rightarrow j}$ as random values in $\{1, \dots, n^4\}$, for every $j \in \{1, \dots, n\}$. P_i then runs the following for $T := \lceil n/t + 1 \rceil$ times in parallel. That is, for $\ell \in [1, \dots, T]$, each P_i acting as the dealer executes the following in parallel:
 - (a) Let the set of moderators be $\mathcal{M}_\ell = (P_{(\ell-1) \cdot (t+1) + 1}, \dots, P_{\ell \cdot (t+1)})$.
 - (b) The dealer P_i chooses a random $(2t, t)$ -bivariate polynomial $S^{i, \ell}(x, y)$ while hiding the $t+1$ values $c_{i \rightarrow j}$ for every $j \in \{(\ell-1) \cdot (t+1) + 1, \dots, \ell \cdot (t+1)\}$, one corresponding to each moderator $P_j \in \mathcal{M}_\ell$. Specifically, P_i chooses $S^{i, \ell}(x, y)$ such that $S^{i, \ell}(0, 0) = c_{i \rightarrow (\ell-1) \cdot (t+1) + 1}$ and so on till $S^{i, \ell}(-t, 0) = c_{i \rightarrow \ell \cdot (t+1)}$. The parties invoke $\mathcal{F}_{\text{mm-pVSS}}$ (Fig. 3.6.1) where P_i is the dealer, and the moderators are parties in \mathcal{M}_ℓ .

(c) Each party P_k gets as output a pair of shares $f_k^{i,\ell}(x), g_k^{i,\ell}(y)$, outputs $d_{i,j}^k$ and a flag $v_{i,j}^k$ for each moderator $P_j \in \mathcal{M}_\ell$.

Note that the above is run for all dealers P_1, \dots, P_n in parallel, where each dealer has T parallel instances (in total $T \cdot n$ invocations).

Upon completion of the above, let succeeded_i be the set of moderators for which P_i holds a flag 1 in all executions, i.e., $\text{succeeded}_i := \{j \mid v_{d,j}^i = 1 \text{ for all dealers } P_d \in \mathcal{P}\}$.

2. **Reconstruct the weights and pick a leader:** The reconstruction phase, $\Pi_{\text{mm-pVSS}}^{\text{Rec}}$ (Fig. 3.6.8) of each of the above nT instances of multi-moderated packed secret sharing is run in parallel to reconstruct the secrets previously shared.

Let $c_{i \rightarrow j}^k$ denote P_k 's view of the value $c_{i \rightarrow j}$ for every $i, j \in \{1, \dots, n\}$, i.e., the reconstructed value for the instance where P_i is the dealer and P_j is the moderator.

Each party P_k sets $c_j^k = \sum_{i=1}^n c_{i \rightarrow j}^k \bmod n^4$ and outputs j that minimizes c_j^k among all $j \in \text{succeeded}_k$ (break ties arbitrarily).

Theorem 3.7.3. *Let $t < n/3$. Protocol Π_{OLE} (Protocol 3.7.2) computes \mathcal{F}_{OLE} (Functionality 3.7.1) in the presence of a malicious adversary corrupting at most t parties. The protocol requires a transmission of $\mathcal{O}(n^4 \log n)$ bits over point-to-point channels.*

Proof. The simulator first simulates $\mathcal{F}_{\text{mm-pVSS}}$ (Functionality 3.6.1) for all the packed secret sharings of all the honest dealers. For this, it generates random shares $f_k^{i,\ell}(x), g_k^{i,\ell}(y)$ for each $\ell \in \{1, \dots, T\}$ on behalf of every honest dealer P_i for the parties corrupted by the adversary. Observe that these shares do not determine the underlying values $c_{i \rightarrow j}$ (see Claim 3.3.3) for each $j \in \{1, \dots, n\}$. Note that each party acts as the moderator in one instance of multi-moderated secret sharing where P_i is the dealer. Hence, for every honest moderator P_j , it sets $d_{i,j}^k = \text{accept}$ and flag $v_{i,j}^k = 1$ for every P_k . For each corrupted moderator P_j , the simulator receives from the adversary, a message m_j as per Functionality 3.6.1, which determines the output $d_{i,j}^k$ and flag $v_{i,j}^k$ for all honest parties P_k and whether the reconstruction would be a default value (i.e., 0 when $d_{i,j}^k = \text{reject}$) or the secret of the dealer otherwise. Following this, the simulator simulates Functionality 3.6.1 for all the packed secret sharings of all corrupted dealers. First, for every corrupt P_i , it receives from the adversary the shares $f_k^{i,\ell}(x), g_k^{i,\ell}(y)$ for each honest P_k and every $\ell \in \{1, \dots, T\}$. For each honest moderator P_j , it sets the flag $v_{i,j}^k = 1$ for every P_k . Further, it receives from the adversary a message m_j as per Functionality 3.6.1. Depending on m_j and whether the shares of the honest parties received from the adversary define a unique $(2t, t)$ polynomial, the simulator sets $d_{i,j}^k$ to accept or reject as per the functionality. Further, as in the case of honest dealers, it receives from the adversary, for each

corrupted moderator, a message as per Functionality 3.6.1, which similarly determines the outputs of all honest parties and whether the reconstruction would be a default value (i.e., 0) or the secret of the dealer. Thus, given the shares of the honest parties, $d_{i,j}^k$ and flag $v_{i,j}^k$, the simulator can fully compute the view of each party P_k of the value $-c_{i \rightarrow j}^k$ for every corrupted P_i , every moderator P_j . Moreover, given the flags $v_{i,j}^k$, the simulator can compute succeed_k set for each every honest P_k .

Given all the above information, the simulator can set the sampling algorithm D as follows (this also defines the family \mathcal{D}). The sampling algorithm D is parameterized with the values $c_{i \rightarrow j}^k$ for every corrupted P_i and every moderator P_j , and the flags $v_{i,j}^k$ for every P_i, P_j . Further, the sampling algorithm uses its randomness r to pick all the secret $c_{i \rightarrow j}$ values for every honest dealer P_i and every moderator P_j . Then, given the values $c_{i \rightarrow j}$ and the corresponding $v_{i,j}^k$ for each P_k , the algorithm can simulate the output $c_{i \rightarrow j}^k$ for each party P_k , i.e., P_k 's view of the value $c_{i \rightarrow j}$ for every honest dealer P_i and moderator P_j . Consequently, it can compute $c_j^k = \sum_{i=1}^n c_{i \rightarrow j}^k \bmod n^4$ and set ℓ_k as the index $j \in \{1, \dots, n\}$ that minimizes c_j^k among all $j \in \text{succeed}_k$, just as the protocol.

Note that the family \mathcal{D} is defined as described in the algorithm above, and therefore to specify D it is enough for the simulator to just send the parameterized values $c_{i \rightarrow j}^k$ for every corrupted P_i , and the flags $v_{i,j}^k$. Therefore the functionality itself is also efficient.

Upon defining the sampling algorithm as above, the simulator sends D to the functionality. The functionality returns the randomness used for sampling, which is essentially all the values $c_{i \rightarrow j}$ corresponding to every honest dealer P_i and every moderator P_j . Using these values, the simulator can then generate the polynomials $S^{i,\ell}(x, y)$ for each $\ell \in \{1, \dots, T\}$ that each honest dealer uses in its T instance of the multi-moderated secret sharing, as a function of the shares it has sent to the adversary so far. That is, the simulator sets $S^{i,\ell}(x, y)$ as described in Step 1b under the constraint that $S^{i,\ell}(x, k) = f_k^{i,\ell}(x)$ and $S^{i,\ell}(k, y) = g_k^{i,\ell}(y)$ sent to the adversary for each corrupt P_k . The simulator then uses these polynomials to simulate the reconstruction phase of each instance of multi-moderated secret sharing. For this, the simulator sends $f_k^{i,\ell}(x)$ for each honest P_k to the adversary, where for an honest P_i , the simulator sets $f_k^{i,\ell}(x) = S^{i,\ell}(x, i)$ using $S^{i,\ell}(x, y)$ computed as described. By construction of the polynomials $S^{i,\ell}(x, y)$ and the sampler D , the result of the reconstruction phase would be exactly the output of $D(r)$, as is the output of all parties in the ideal execution. Below, we show that D is a valid sampler.

Proving that D is valid. We next show that D is a valid sampler, namely, that for a random r , $D(r)$ outputs the index of an honest party with some probability $\delta > 1/2$. The proof is almost verbatim from [82].

Towards that end, define:

$$\text{succeeded} = \bigcup_{k \in H} \text{succeeded}_k ,$$

where H is the set of all parties that were honest at the end of phase 1 of the protocol. Recall that by the guarantees of multi-moderated secret sharing, even if a single honest party P_k holds $v_{d,j}^k = 1$ then in the execution where P_d is the dealer and P_j is the moderator, the honest parties would have an agreement on the reconstructed value.

Hence, by the properties of multi-moderated secret sharing, if reconstruction is successful and if $k \in \text{succeeded}$, then for any honest P_i, P_j and any $1 \leq \ell \leq n$ we have that $c_{\ell \rightarrow k}^i = c_{\ell \rightarrow k}^j$, i.e., the outputs of P_i and P_j are the same in the reconstruction associated with the instance with P_ℓ as a dealer and P_k as the moderator. As a result, we can omit the superscripts i and j .

We claim that all values c_k for $k \in \text{succeeded}$ are uniformly distributed in $\{1, \dots, n^4\}$. Note that the set succeeded might contain parties that are controlled by the adversary, but acted honestly while moderating the sharing phases and therefore are also considered. Consider the value $c_{i \rightarrow k}$, that is, the instance of multi-moderated secret sharing where P_i is the dealer and P_k is the moderator. We have the following cases to consider:

1. The moderator P_k is honest, then all the honest parties see the same values $c_{1 \rightarrow k}, \dots, c_{n \rightarrow k}$ regardless of whether each respective dealer P_1, \dots, P_n is honest or not. Moreover, k is in the set succeeded_ℓ of all honest parties P_ℓ . Furthermore,
 - (a) If the dealer P_i is honest then $c_{i \rightarrow k}$ is uniformly distributed in $\{1, \dots, n^4\}$. Moreover, the shares received by the corrupted parties are independent of $c_{i \rightarrow k}$.
 - (b) If the dealer P_i is corrupted then $c_{i \rightarrow k}$ must have been chosen independently of all other values, due to the secrecy property of secret sharing.
2. The moderator P_k is corrupted, then all the honest parties see the same values $c_{1 \rightarrow k}, \dots, c_{n \rightarrow k}$ regardless of whether each respective dealer P_1, \dots, P_n is honest or not. However, k might not be in the set succeeded_ℓ of all honest parties P_ℓ . Furthermore,
 - (a) If the dealer P_i is honest then $c_{i \rightarrow k}$ is uniformly distributed in $\{1, \dots, n^4\}$.
 - (b) If the dealer P_i is corrupted then $c_{i \rightarrow k}$ must have been chosen independently of all other values.

We therefore conclude that all the c_k for $k \in \text{succeeded}$ are distributed uniformly at random in $\{1, \dots, n^4\}$. Let HonestChosen be the event where the index k for which c_k is minimal among

all parties in succeeded is an index of an honest party. We have that:

$$\begin{aligned}
& \Pr [\text{HonestChosen}] \\
& \geq \Pr [\text{HonestChosen} \mid \forall i, j \in \text{succeeded } c_i \neq c_j] \cdot \Pr [\forall i, j \in \text{succeeded } c_i \neq c_j] \\
& \geq \frac{n-t}{n} \cdot (1 - \Pr [\exists i, j \in \text{succeeded}, c_i = c_j]) \\
& \geq \frac{n-t}{n} \cdot \left(1 - n^2 \cdot \frac{1}{n^4}\right) \geq \frac{n-t}{n} - \frac{1}{n^2} \geq \frac{1}{2}.
\end{aligned}$$

□

3.8 Broadcast

3.8.1 Byzantine Agreement

In a Byzantine agreement, every party P_i holds initial input v_i and the following properties hold: **(Agreement)**: All the honest parties output the same value; **(Validity)**: If all the honest parties begin with the same input value v , then all the honest parties output v . We simply plug in our OLE in the Byzantine agreement of [82]. As described in Section 3.1.3, we present standalone functionalities for Byzantine agreement and broadcast, where the intricacies of sequential composition are tackled in [54].

Functionality 3.8.1: \mathcal{F}_{BA} – Byzantine Agreement

The functionality is parameterized by the set of corrupted parties I .

1. The functionality receives from each honest party P_j its input $b_j \in \{0, 1\}$. The functionality sends $(b_j)_{j \notin I}$ to the adversary.
 2. The adversary sends a bit \hat{b} .
 3. If there exists a bit b such that $b_j = b$ for every $j \notin I$, then set $y = b$. Otherwise, set $y = \hat{b}$.
 4. Send y to all parties.
-

The protocol for byzantine agreement appears below, followed by the proof of its security.

Protocol 3.8.2: Π_{BA} – Byzantine Agreement Protocol

Input: Each party P_i holds a bit b_i .

Initialization: Each party initializes `decidedi = false` and `openToAcceptRandom = false`. Run the following iteratively until termination:

1. **Round I – each party P_i :**
 - (a) Send b_i to all parties.
 - (b) Let $b_{j,i}$ be the bit received from P_j (if no value was received, use the value from the previous iteration; at the outset of the protocol, use a default value).
 2. **Round II – each party P_i :**
 - (a) Set $\mathcal{S}_i^0 := \{j \mid b_{j,i} = 0\}$ and $\mathcal{S}_i^1 := \{j \mid b_{j,i} = 1\}$.
 - (b) If $|\mathcal{S}_i^0| \geq t + 1$ then set $b_i = 0$. If $|\mathcal{S}_i^0| \geq n - t$ then set $\text{decided}_i = \text{true}$.
 - (c) Send b_i to all parties. If a value was received from party P_j , then store it as $b_{j,i}$.
 3. **Round III – each party P_i :**
 - (a) Update \mathcal{S}_i^0 and \mathcal{S}_i^1 according to the new values $b_{1,i}, \dots, b_{n,i}$.
 - (b) If $|\mathcal{S}_i^1| \geq t + 1$ then set $b_i = 1$. If $|\mathcal{S}_i^1| \geq n - t$ then set $\text{decided}_i = \text{true}$.
 - (c) Send b_i to all parties. If a value was received from party P_j , then store it as $b_{j,i}$.
 4. **Round IV – each party P_i :**
 - (a) If $\text{decided}_i = \text{false}$ then set $\text{openToAcceptRandom}_i = \text{true}$.
 - (b) Update \mathcal{S}_i^0 and \mathcal{S}_i^1 according to the new values $b_{1,i}, \dots, b_{n,i}$.
 - (c) If $|\mathcal{S}_i^0| \geq t + 1$ then set $b_i = 0$. If $|\mathcal{S}_i^0| \geq n - t$ then set $\text{openToAcceptRandom}_i = \text{false}$.
 - (d) Send b_i to all parties. If a value was received from party P_j , then store it as $b_{j,i}$.
 5. **Round V – each party P_i :**
 - (a) Update \mathcal{S}_i^0 and \mathcal{S}_i^1 according to the new values $b_{1,i}, \dots, b_{n,i}$.
 - (b) If $|\mathcal{S}_i^1| \geq t + 1$ then set $b_i = 1$. If $|\mathcal{S}_i^1| \geq n - t$ then set $\text{openToAcceptRandom}_i = \text{false}$.
 - (c) Send b_i to all parties. If a value was received from party P_j , then store it as $b_{j,i}$.
 6. **Round VI – each party P_i :**
 - (a) All parties execute Π_{OLE} (Protocol 3.7.2) and let ℓ_i be the output of P_i .
 - (b) If $\text{openToAcceptRandom}_i = \text{true}$, then set $b_i = \ell_i$.
 - (c) If $\text{decided}_i = \text{true}$, then output b_i and terminate. Otherwise, proceed to the next iteration.
-

Theorem 3.8.3. *Protocol Π_{BA} (Protocol 3.8.2) is a Byzantine agreement protocol tolerating t malicious parties that works in constant expected rounds and requires the transmission of $\mathcal{O}(n^4 \log n)$ bits in expectation for $n \geq 3t + 1$.*

Proof. Simulation is straightforward: since the simulator receives all the inputs of the honest parties it can perfectly simulate them. Moreover, the simulator can also perfectly simulate the \mathcal{F}_{OLE} . It just receives a (valid) sampler algorithm from the adversary, runs it, and gives the randomness it used to the adversary, while also receiving the outputs of each honest party. Thus, the view of the adversary is identical between the real and ideal executions.

The simulator then sees the output of the simulated honest parties \hat{b} , and send that bit \hat{b} to the trusted party. We will next show that all simulated honest parties must output the same bit (this is essentially the “agreement” property). The trusted party then decides what to send to the honest parties. If all honest parties sent the same input b to the trusted party, then it ignores the bit that the simulator had sent it and just output b (this is essentially the “validity” property). Otherwise, it outputs \hat{b} .

We now show that the protocol satisfies agreement and validity. This in particular shows that the output of the simulated honest parties in the simulated execution is the same as the output of the honest parties in the ideal execution. Moreover, this also implies that the output in the ideal execution is identical to the real.

The proof of agreement and validity here is taken almost verbatim from [82]. The proofs of the following properties can be found in [82] and we give them here for completeness.

Claim 3.8.4. *Protocol Π_{BA} (Protocol 3.8.2) satisfies the following properties:*

Property I: *If at the beginning of some iteration, all (remaining) honest parties P_i hold the same value $b_i = b$, then all honest parties who have not yet terminated will output b and terminate the protocol at the end of the iteration.*

Property II: *If some party P_i sets $\text{decided}_i = \text{true}$ at some iteration, then by the end of that iteration, each honest party P_j that has not yet terminated holds $b_j = b_i$, regardless of the result of the OLE protocol.*

Property III: *If an honest party P_i sets $\text{openToAcceptRandom}_i = \text{false}$ in some iteration and holds a bit $b_i = b$, then all honest parties that have not yet terminated hold $b_j = b_i = b$ by the end of Round V of that iteration.*

Property IV: *If some party P_i terminates with output $b_i = b$, then all honest parties terminate with identical output in either the current iteration or in the next one, regardless of the results of the OLE protocol.*

Property V: *If an honest leader P_ℓ is elected in Round VI of some iteration, then all honest parties P_i terminate by the end of the next iteration.*

Property I. Assume that $b_i = b = 0$ at the beginning of the iteration for every honest party P_i . Then, it holds that $|\mathcal{S}_i^0| \geq n - t$ and hence P_i sets $\text{decided}_i = \text{true}$ at Step 2b. This implies

that b_i cannot be changed in Round III and remains 0. Consequently, in Round IV, P_i sets $\text{openToAcceptRandom}_i = \text{false}$ and b_i stays 0 (as $|\mathcal{S}_i^0| \geq n - t$). Further, b_i remains unchanged in Round V (since $|\mathcal{S}_i^1| \leq t$). Finally, in Round VI the parties run the OLE protocol, but ignore its value since $\text{openToAcceptRandom}_i = \text{false}$. Since $\text{decided}_i = \text{true}$, every honest P_i outputs $b_i = 0$ and terminates.

The case where all parties start with $b_i = b = 1$ is shown analogously.

Property II. Assume that P_i sets $\text{decided} = \text{true}$ at Step 2b. This implies that $|\mathcal{S}_i^0| \geq n - t$. Since at most t parties from \mathcal{S}_i^0 can be corrupt, for every other honest party P_j it holds that $||\mathcal{S}_i^0| - |\mathcal{S}_j^0|| \leq t$ and thus $|\mathcal{S}_j^0| \geq n - 2t \geq t + 1$. Hence, at the end of Step 2b every honest party P_j sets $b_j = b_i = 0$. All honest parties then send their new bits in Step 2c, and thus at Round III we have that $|\mathcal{S}_j^1| \leq t$ for every honest P_j and therefore b_j remains 0. As a result, in Round IV all honest parties set $\text{openToAcceptRandom}_j = \text{false}$, and b_j remains 0 for all honest parties at the end of Round V. In Round VI, parties run the OLE protocol, but ignore its value since $\text{openToAcceptRandom}_i = \text{false}$.

Consequently, in the next round, all honest parties (that did not terminate) start with the same input, and as follows from Property I, all terminate with the same value as the output. An analogous argument can be shown for the case when P_i sets $\text{decided}_i = \text{true}$ at Step 3b.

Property III. A party P_i sets $\text{openToAcceptRandom}_i = \text{false}$ if $|\mathcal{S}_i^0| \geq n - t$ in Step 4c. As shown in the proof of the previous property, this implies that for every other honest party P_j it holds that $|\mathcal{S}_j^0| \geq t + 1$, and thus P_j sets $b_j = 0$ (although it might keep the flag $\text{openToAcceptRandom}_i = \text{true}$ if $|\mathcal{S}_j^0| < n - t$). The value b_j does not change during round V, from a similar reasoning as in the previous claim.

A similar argument holds for the case when P_i sets $\text{openToAcceptRandom}_i = \text{false}$ in Step 5b.

Property IV. A party P_i terminates only when $\text{decided}_i = \text{true}$. Property II shows that all other honest parties P_j would hold $b_j = b_i = b$ at the end of the iteration, while some might terminate. Further, by virtue of Property I, all the honest parties which do not terminate at the end of the iteration are guaranteed to terminate by the end of the next iteration.

Property V. When an honest leader P_ℓ is elected in Round VI, every honest party P_j obtains the same value $\ell_j = \ell$ as the output of Π_{OLE} . Moreover, all the honest parties must have received the same bit b_ℓ from an honest P_i in the prior rounds. If all the honest parties P_j hold $\text{openToAcceptRandom}_j = \text{true}$, then all set $b_j = b_\ell$, and thus begin the next iteration with the same value. By virtue of Property I, this implies that all honest parties output b_ℓ by the end of the next iteration. Otherwise, if some honest party P_i has $\text{openToAcceptRandom}_i = \text{false}$,

then due to Property III it holds that $b_\ell = b_i$ by the end of Round V. Thus, every other honest party P_j sets $b_j = b_\ell = b_i$ in Round VI. Similar to the prior case, this implies that all the (remaining) honest parties begin the next iteration with the same value and hence output b_ℓ by the end of the next iteration.

It is guaranteed that the OLE protocol (Π_{OLE} , Protocol 3.7.2) elects an honest leader with constant probability as shown in Theorem 3.7.3. It thus follows that agreement is reached in expected number of iterations, where each iteration requires only a constant number of rounds.

Efficiency. In each iteration the parties send $O(n^2)$ bits over the point-to-point channels, and then run OLE protocol (Π_{OLE} , Protocol 3.7.2), which requires $O(n^4 \log n)$ bits of communication over point-to-point channels. \square

3.8.2 Broadcast and Parallel-broadcast

In a broadcast protocol, a distinguished dealer $P^* \in \mathcal{P}$ holds an initial input M and the following hold: **(Agreement):** All honest parties output the same value; **(Validity):** If the dealer is honest, then all honest parties output M . We formalize it using the following functionality:

Functionality 3.8.5: \mathcal{F}_{BC}

The functionality is parametrized with a parameter L .

1. The dealer (sender) P^* sends the functionality its message $M \in \{0, 1\}^L$.
 2. The functionality sends to all parties the message M .
-

To implement this functionality, the dealer just gradecasts its message M and then parties run Byzantine agreement on the grade they received, while parties use input 1 for the Byzantine agreement if and only if the grade of the gradecast is 2. If the output of the Byzantine agreement is 1, then they output the message they received in the gradecast, and otherwise, they output \perp . We simply plug in our gradecast and Byzantine agreement in the above protocol. Note that the above communication complexity is asymptotically free (up to the expectation) for $L > n^3 \log n$.

Protocol 3.8.6: Π_{BC} – Broadcast Protocol for a single dealer

- **Input:** The dealer holds a message $M \in \{0, 1\}^L$.
- **Common input:** A parameter L .
 1. **The dealer:** Gradecast M .

2. **Each party P_i :** Let M' be the resultant message and let g be the associated grade. All parties run Byzantine agreement where the input of P_i is 1 if $g = 2$, and otherwise the input is 0.
- **Output:** If the output of the Byzantine agreement is 1 then output M' . Otherwise, output \perp .
-

Theorem 3.8.7. *Protocol 3.8.6 is a secure broadcast tolerating $t < n/3$ malicious parties. For an input message M of length L bits, the protocol requires $\mathcal{O}(nL)$ plus expected $\mathcal{O}(n^4 \log n)$ bits total communication, and expected constant number of rounds.*

Proof. We prove the protocol in the $\mathcal{F}_{\text{BA}}\text{-}\mathcal{F}_{\text{Gradecast}}$ hybrid model.

The case of a corrupted sender. In case of a corrupted sender, the simulator simulates the $\mathcal{F}_{\text{Gradecast}}$ functionality and receives from the adversary either:

1. $(\text{ExistsGrade2}, M, (g_j)_{j \notin I})$. Then, simulate the \mathcal{F}_{BA} functionality where all honest parties input either 1 or 0 according to $(g_j)_{j \notin I}$, where note that $\mathcal{F}_{\text{Gradecast}}$ guarantees that all $g_j \geq 1$ and there exists at least one index for which $g_j = 2$. The \mathcal{F}_{BA} sends to the adversary all the bits of the honest parties and then receives back one bit, \hat{b} . If $g_j = 2$ for every j , or if $\hat{b} = 1$, then the simulator sends M to \mathcal{F}_{BC} . Otherwise, it sends \perp to \mathcal{F}_{BC} . \mathcal{F}_{BC} forwards the chosen message to all parties.
2. $(\text{NoGrade2}, (M_j)_{j \notin I}, (g_j)_{j \notin I})$. This time, it is guaranteed that all honest parties have $g_j \leq 1$. Then, simulate the \mathcal{F}_{BA} functionality where all honest parties input 0 to \mathcal{F}_{BA} . This implies that the output of \mathcal{F}_{BA} is \perp to all parties. The simulator then sends \perp to \mathcal{F}_{BC} , which forwards that message to all parties.

From inspection, the view of the adversary in the real and ideal is identical. Likewise, the output of the honest parties.

The case of an honest sender. In this case, the simulator receives M from the trusted party. It then simulates the $\mathcal{F}_{\text{Gradecast}}$ sending $(M, 2)$ to all corrupted parties. Next, it simulates \mathcal{F}_{BA} , considering all honest parties send 1 to \mathcal{F}_{BA} . It receives some bit \hat{b} from the adversary which it ignores as its input to \mathcal{F}_{BA} , and simulates the output of \mathcal{F}_{BA} to be 1.

From inspection, it is easy to see that the joint distribution of the view of the adversary and the outputs of the honest parties in the real is identically distributed to the view and the outputs in the ideal.

Efficiency. The protocol gradecasts a message which requires $O(nL)$ bits of communication and runs in constant rounds. In addition, we run Byzantine agreement, which requires expected $O(n^4 \log n)$ bits of communication in expected constant rounds.

□

Parallel Broadcast. Parallel broadcast relates to the case where n parties wish to broadcast a message of size L bits in parallel. In that case, we rely on an idea of Fitzi and Garay [70] that applies to OLE-based protocols. The idea is that the multiple broadcast sub-routines are run in parallel when only a single election per iteration is required for all these sub-routines. This results in the following corollary:

Corollary 3.8.8. *There exists a perfectly secure parallel-broadcast with optimal resilience, which allows n parties to broadcast messages of size L bits each, at the cost of $\mathcal{O}(n^2L)$ bits communication, plus $\mathcal{O}(n^4 \log n)$ expected communicating bits. The protocols runs in constant expected number of rounds.*

For completeness, we provide the functionality for parallel broadcast below, and omit the proof since it follows from broadcast.

Functionality 3.8.9: $\mathcal{F}_{\text{BC}}^{\text{parallel}}$

The functionality is parametrized with a parameter L .

1. Each $P_i \in \mathcal{P}$ sends the functionality its message $M_i \in \{0, 1\}^L$.
 2. The functionality sends to all parties the message $\{M_i\}_{i \in \{1, \dots, n\}}$.
-

Efficiency. The protocol gradecasts n messages, each of which requires $O(nL)$ bits of communication and runs in constant rounds. In addition, we run Byzantine agreement where a single leader election per iteration is necessary across all the instances, which requires expected $O(n^4 \log n)$ bits of communication in expected constant rounds.

Chapter 4

Detect, Pack and Batch: Perfectly-Secure MPC with Linear Communication and Constant Expected Time

In this chapter, we discuss our construction of the perfectly-secure MPC protocol with optimal resilience in the synchronous network. Along the way, we provide an improved construction for verifiable secret sharing and introduce a new primitive which we refer to as *detectable secret sharing*.

4.1 Introduction

We consider the most demanding setting: perfect security with optimal resilience. A noted earlier, perfect security means that the adversary is all-powerful and that the protocol has zero probability of error. Optimal resilience means that the number of corruptions is at most $t < n/3$ [85, 94, 29].

As described before, the seminal protocols of Ben-Or, Goldwasser, and Wigderson [29], and Chaum, Crépeau and Damgård [46] led the foundations of this setting. Since then, there are, in general, two families of protocols:

1. **Efficient but slow:** These protocols [80, 26, 76] ([26] test-of-time award) have $\mathcal{O}(n \log n)$ communication complexity per multiplication gate. Still, the running time of these protocols is at least $\Theta(n)$ rounds, even if the depth of the circuit is much smaller $D \ll n$. Specifically:

Theorem 4.1.1. *For an arithmetic circuit with C multiplication gates and depth D there exists a perfectly-secure, optimally-resilient MPC protocol with $\mathcal{O}(n^5 \log n + Cn \log n)$ bits communication complexity and $\Omega(n + D)$ expected number of rounds.*

The protocol requires $\mathcal{O}(n^3 \log n + Cn \log n)$ bits of point-to-point communication and n sequential invocations of broadcast of $\mathcal{O}(\log n)$ bits each, with $\Omega(n + D)$ rounds. Using the broadcast implementation of [7], this becomes the complexity of Theorem 4.1.1. Alternatively, using the implementation of [33, 53], the protocol can be more efficient, but even more slower: $\mathcal{O}(n^3 \log n + Cn \log n)$ bits communication complexity and $\Omega(n^2 + D)$ number of rounds.

2. **Fast but not efficient:** This line of protocols [29, 46, 73, 56, 19, 6] run at $\mathcal{O}(D)$ expected number of rounds, but require $\Omega(n^4 \log n)$ communication complexity per multiplication gate.

Theorem 4.1.2. *For an arithmetic circuit with C multiplication gates and depth D there exists a perfectly-secure, optimally-resilient MPC protocol with $\Omega(Cn^4 \log n)$ communication complexity and $\mathcal{O}(D)$ expected number of rounds.*

In the broadcast hybrid model, the protocol requires $\mathcal{O}(n^3 \log n)$ bits of communication complexity over point-to-point channels and $\mathcal{O}(n^3 \log n)$ bits broadcast, in $\mathcal{O}(D)$ number of rounds. Theorem 4.1.2 reports the communication complexity using the broadcast implementation of [7]. Using [33, 53] for implementing the broadcast, the number of rounds is increased to $\Omega(n + D)$.

Our Main Result

Our main result is that it is possible to simultaneously achieve the best of both families. For the first time, we provide a perfectly-secure, optimally-resilient MPC protocol that has **both** $\mathcal{O}(n \log n)$ communication complexity per multiplication gate and $\mathcal{O}(D)$ expected time complexity.

Theorem 4.1.3 (Main Result). *For a circuit with C multiplication gates and depth D there exists a perfectly-secure, optimally-resilient MPC protocol with $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$ communication complexity and $\mathcal{O}(D)$ expected number of rounds.*

In the broadcast-hybrid model, the total communication complexity over point-to-point is $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$, and each party has to broadcast at most $\mathcal{O}(n^2 \log n)$ bits. Using [7] for implementing the broadcast, we obtain Theorem 4.1.3. Compared to [26, 76], for $D \ll n$, our result provides up to an $\mathcal{O}(n)$ improvement in round complexity while keeping the same linear communication complexity (and also improving the communication complexity for $C \in o(n^4)$). Compared to [6], for $C > n^3$, our result provides an $\mathcal{O}(n^3)$ improvement in the communication complexity while keeping the same $\mathcal{O}(D)$ expected round complexity.

We remark that in many practical settings, a large set of parties may want to compute a shallow depth circuit in a robust manner. For instance, consider a network with 200ms latency and channels of 1Gbps, and consider a highly parallel circuit with 1M gates, depth $D = 10$, and $n = 200$ parties. Then, the round complexity of our protocol is $\mathcal{O}(D)$, which results in a delay of $10 \cdot 200\text{ms} = 2$ seconds. The delay associated with the communication complexity is smaller: each party sends or receives $(C + Dn + n^3) \log n$ bits, which over 1Gbps channel results in a delay of 0.08 seconds. In [76], the delay due to the round complexity is $\mathcal{O}(n+D)$, which results in a delay of $210 \cdot 200\text{ms} = 42$ seconds, and each party sends or receives $(C + n^4) \log n$ bits which over 1Gbps results in a delay of ≈ 14 seconds. If we use [33, 53] to implement the broadcast, then the round complexity becomes $\mathcal{O}(n^2 + D)$ which is ≈ 8000 seconds. The improvement in the round complexity is significant in this scenario. Of course, these are only coarse estimations that do not even take into account the hidden constants in the \mathcal{O} notation.

Main Technical Result

Our main result is obtained via several advances in building blocks for perfectly secure optimally resilient MPC. In our view, the most important and technically involved contribution is a new primitive called *Detectable Secret Sharing*. This is a secret sharing with the following properties: (1) Secrecy: The corrupted parties cannot learn anything about the secrets after the sharing phase for an honest dealer; (2) Binding: After the sharing phase (even if the dealer is corrupted), the secret is well defined by the shares of the honest parties; (3) Reconstruction or detection: Reconstruction ends up in the well-defined secret, or it might fail (even if the dealer is honest). However, in the case of failure, there is a (private) detection of $\mathcal{O}(n)$ corrupted parties. Moreover, successful sharing and reconstruction are guaranteed if the dealer has already detected more than $t/2$ corrupted parties before the respective phases.

We show that despite the possible failure of the reconstruction, Detectable Secret Sharing suffices for obtaining our end result for MPC. Most importantly, we obtain a highly efficient construction for this primitive:

Theorem 4.1.4 (informal). *There exists a detectable secret sharing protocol that allows sharing p secrets (of $\log n$ bits each) with malicious security and optimal resilience with $\mathcal{O}(n^4 \log n + p \log n)$ communication complexity and expected constant number of rounds.*

For $p \geq n^4$, this is $\mathcal{O}(1)$ field elements per secret (which is also a field element)! This matches packed semi-honest secret sharing as in [71]. The theorem holds for a single dealer; for n dealers, each sharing p secrets in parallel, we get $\mathcal{O}(1)$ field elements per secret starting from $p \geq n^3$.

Stated differently, we show a detectable secret sharing protocol that can pack $\mathcal{O}(n^2)$ secrets (of size $\log n$ each) at the cost of $\mathcal{O}(n^2 \log n)$ communication complexity for private channels and each party broadcasts at most $\mathcal{O}(n \log n)$ bits, with a strictly constant number of rounds. There are at least two striking features of our new detectable secret sharing: *packing*, and *batching*. First, to the best of our knowledge, this is the first protocol in the malicious setting that can pack $\mathcal{O}(n^2)$ secrets at the cost of $\mathcal{O}(n^2)$ communication complexity – so an amortized cost of $\mathcal{O}(1)$ per secret over point-to-point channels. Second, our scheme allows batching – m independent instances with the same dealer require $\mathcal{O}(mn^2 \log n)$ over point-to-point channels but just $\mathcal{O}(n \log n)$ broadcast per party in all m instances combined. To the best of our knowledge, this is the first protocol that requires a fixed broadcast cost independent of the batching parameter m . By setting $m = p/n^2$ and combining with the recent broadcast implementation of Abraham, Asharov, Patil, and Patra [7], we obtain Theorem 4.1.4 in the point-to-point channel model and no broadcast.

Note that this primitive is formally incomparable with weak-secret sharing [98] (where reconstruction needs the help of the dealer but is guaranteed to succeed when the dealer is honest). On the one hand, our notion seems weaker as there is no guaranteed validity (no guaranteed reconstruction in case of an honest dealer). On the other hand, it is not strictly weaker since our notion ensures mass detection in case of a reconstruction failure. For comparison, the best known weak-secret sharing [6] requires $\mathcal{O}(n^4 \log n)$ for sharing $\mathcal{O}(n)$ secrets (i.e., $\mathcal{O}(n^3)$ per secret).

Verifiable secret sharing. We also derive (and use) a “strong” secret sharing (i.e., honest parties always succeed to reconstruct), i.e., in the standard verifiable secret sharing [48] setting:

Theorem 4.1.5 (informal). *There exists a protocol that allows to secret share p secrets (of $\log n$ bits each) with malicious security and optimal resilience with $\mathcal{O}(n^4 \log n + p \cdot n \log n)$ communication complexity and expected constant number of rounds.*

For $p \geq n^3$, this is an overhead of $\mathcal{O}(n)$ per secret. Previously, the best known [7] in this setting packs $\mathcal{O}(n)$ secrets with $\mathcal{O}(n^4 \log n)$ communication complexity (an overhead of $\mathcal{O}(n^3)$ per secret). This is an improvement of $\mathcal{O}(n^2)$ over the state-of-the-art. In comparison, the starting point is the VSS of BGW and Feldman [29, 67] requires $\mathcal{O}(n^2 \log n)$ point-to-point and $\mathcal{O}(n^2 \log n)$ broadcast, for sharing just a single secret. This results in $\mathcal{O}(n^4 \log n)$ communication complexity over point-to-point channels and no broadcast, for sharing just a single secret (an overhead of $\mathcal{O}(n^4)$).

Detection. The line of work of [80, 26, 76] in perfectly-secure MPC is based on the *player elimination framework* (introduced by Hirt, Maurer and Przydatek [80]). The protocol identifies a set of parties in which it is guaranteed that one of the players among the set is corrupted, excludes the entire set, and restarts the protocol. The important aspect here is that all parties agree on the set, and that honest parties are also “sacrificed” along the way. In each iteration, the number of parties being excluded is constant. This is a slow process that leads to the $\mathcal{O}(n)$ rounds overhead.

Instead of globally eliminating a set of parties, our approach is to have each party maintain a local set of conflicted parties, with no global agreement among parties on who is malicious. Each party can decide which parties to mark as conflicted while it shares its own secret(s). When an honest party marks enough corrupt parties as conflicted, its sharing will always be successful. Moreover, whenever there is a failure in sharing or reconstruction, then there is a mass detection – $\mathcal{O}(n)$ corruptions are identified, either publicly or privately.

To elaborate further, our MPC protocol uses three kinds of detections:

1. *Global detection* – wherein a set of parties is excluded from the computation. Unlike [80, 26, 76], in our case, honest parties are never discarded (e.g., “discard the dealer”).
2. *Public individual detection* – wherein each party has its own conflict set that is publicly known to all (see, e.g., Step 2b in Protocol 4.4.2). While a similar mechanism, referred to as ‘dispute control’ has been used in [24, 32, 77], these works achieve *statistical* security in the honest majority setting with $\mathcal{O}(n)$ rounds overhead similar to the player-elimination framework;
3. *Private (local) detection* – wherein each party has its private conflict set that it excludes from its local computation. Specifically, an honest party may locally identify a set conflicts (with corrupted parties) without a mechanism to prove that it has done so honestly. In our protocol, it can identify $\mathcal{O}(n)$ such conflicts simultaneously in case private reconstruction towards it fails (see, e.g., Step 7 in Protocol 4.5.2). This allows an honest party to locally discard the communication from $\mathcal{O}(n)$ corrupt parties, eventually

ensuring a successful reconstruction.

4.1.1 Related Work

Linear communication MPC. Achieving (near) linear communication complexity for MPC, has been the target for a long line of works in the literature of information-theoretic MPC with optimal resilience. In the closely-related setting of *statistical* security with a *synchronous* network, the seminal work of [98] established the feasibility of MPC in this setting with the optimal threshold of $t < n/2$, assuming the availability of a broadcast channel in addition to the point-to-point channels. Subsequently, several works focused on the efficiency of protocols, starting from the work of [55] that requires $\mathcal{O}(n^5)$ bits of communication over point-to-point channels as well as broadcast per multiplication gate. A protocol with $\mathcal{O}(n^2)$ cost per gate was known since the work of [24] in 2006 with the optimal threshold of $t < n/2$. In 2012, Ben-Sasson, Fehr, and Ostrovsky [32] provided the first protocol with linear communication per gate (up to $\log n$ factor). Their result is based on ideas inspired by key techniques from PCP literature. Similar to our construction, their protocol additionally had a depth-dependent term that was quadratic in the number of parties. This factor was later eliminated in [77] while maintaining the linear cost per gate.

Broadcast. Our communication complexity takes into account the cost of broadcast. In the setting of perfect security, there are two families of protocols for implementing the broadcast: once again – efficient and slow, or fast but less efficient. The former [33, 53] takes $\mathcal{O}(n)$ rounds and $\mathcal{O}(n^2 + pn)$ for broadcasting a message of p bits. The latter [7] (built upon Feldman and Micali [66], and Katz and Koo [82]) takes $\mathcal{O}(1)$ expected number of rounds and $\mathcal{O}(n^4 + pn)$ communication complexity for broadcasting a message of size p bits, i.e., this is optimal for $p > n^3 \log n$. Note that when broadcasting a message of size p , then since each party is supposed to receive p bits, the minimal possible communication complexity is pn . Moreover, n parties broadcasting messages of size p bits each takes $\mathcal{O}(n^4 + pn^2)$, i.e., optimal for $p > n^2 \log n$. We also remark that containing strict $\mathcal{O}(1)$ number of rounds is impossible [69].

Shunning. Our notion of detectable secret sharing can be viewed as a synchronous analog of the notion of shunning, in which parties either succeed in their asynchronous verifiable secret sharing or some detection event happens. In the context of asynchronous verifiable secret sharing, shunning was first suggested by Abraham, Dolev, and Halpern [3] and later improved and extended to shunning $\mathcal{O}(n)$ parties by Bangalore, Choudhury, and Patra [21, 22]. However, unlike our detectable secret sharing, none of these works attain $\mathcal{O}(1)$ amortized

communication cost per secret.

4.2 Technical Overview

In this section, we provide a technical overview of our work. We start in Section 4.2.1 with an overview of our main technical result – our detectable and verifiable secret sharing schemes. In Section 4.2.2 we overview our MPC result. Most of the building blocks are based on previous works, and we highlight in the overview the steps where we made significant improvements. In Section 4.2.3 we overview another step in the protocol, triplet secret sharing.

4.2.1 Detectable and Verifiable Secret Sharing

We start this overview with the most basic verifiable secret sharing protocol – the one by BGW [29]. See also [18, 11] for further details. To share a secret s , the dealer chooses a bivariate polynomial $S(x, y) = \sum_{k=0}^t \sum_{\ell=0}^t s_{k,\ell} \cdot x^k y^\ell$ of degree t in both x and y under the constraint that $S(0, 0) = s_{0,0} = s$. The share of each party P_i is the pair of degree- t univariate polynomials $S(x, i), S(i, y)$. The goal of the verification step is to verify that the shares of all honest parties indeed lie on a unique bivariate polynomial $S(x, y)$. Let us briefly recall the sharing phase:

1. **Sharing:** The dealer sends the share $(f_i(x), g_i(y)) = (S(x, i), S(i, y))$ to each party P_i .
2. **Pairwise checks:** P_i sends to each P_j the two points $(f_i(j), g_i(j)) = (S(j, i), S(i, j)) = (g_j(i), f_j(i))$. If P_i did not receive from P_j the points it expects to see (i.e., that agree with $f_i(x), g_i(y)$), then it publicly broadcasts a complaint $\text{complaint}(i, j, f_i(j), g_i(j))$.
3. **Publicly resolving the complaints:** The dealer checks all complaints; if some party P_i publicly complains with values that are different than what the dealer has sent it, then the dealer makes the share of P_i public – i.e., it broadcasts $\text{reveal}(i, S(x, i), S(i, y))$.
4. If a party P_j sees that (1) all polynomials that the dealer made public agree with its private shares; (2) its share was not made public; (3) if two parties P_k and P_ℓ both complaint on each other, then the dealer must open one of them. If all those conditions hold, then P_j is happy with its share, and votes to accept the dealer. If the shares of P_j were made public, then it re-assigns $f_j(x), g_j(y)$ to the publicly revealed ones.
5. If $2t + 1$ parties votes to accept the shares, then each party output its share. Otherwise, the dealer is discarded.

Observe that if the dealer is honest, then during the verification phase the corrupted parties do not learn anything new. Specifically, a party always broadcasts a complaint with the values that it received from the dealer, and the dealer makes a share public only if the public

complaint does not contain the values that the dealer has sent that party. Therefore, an honest dealer never makes the shares of another honest party public. Moreover, all honest parties are happy, and accept the shares.

If the dealer is corrupted, then $2t + 1$ parties that voted to accept the dealer implies that we have a set $J \subseteq [n]$ of at least $t + 1$ honest parties that are happy with their shares and that their shares were never made public. The shares of those $t + 1$ honest parties fully determine a bivariate polynomial of degree- t in both variables. If some honest party P_j initially held a share that does not agree with this bivariate polynomial, i.e., does not agree with some P_k for $k \in J$, then it must be that P_j and P_k both publicly complained, and that the share of P_j was made public with some new share that agrees with S (if it does not agree with S , then at least one party in J would have not voted to accept). Therefore, at the end, all honest parties hold shares of a well-defined bivariate polynomial.

To reconstruct the bivariate polynomial, each party sends to each other party its pair of polynomials. Since the underlying polynomial is of degree- t , the adversary controls at most t parties, we must have $n - t \geq 2t + 1$ correct points and at most t errors. The Reed-Solomon decoding procedure guarantees that the t errors can be identified and corrected.

Our improvements. The above scheme for verifiable secret sharing requires $\mathcal{O}(n^2 \log n)$ communication over the point-to-point channels, and also the broadcast of $\mathcal{O}(n^2 \log n)$ bits. This results in total communication complexity of $\mathcal{O}(n^4 \log n)$ over point-to-point for sharing a single secret. The work of [7] has the same complexity for sharing $\mathcal{O}(n)$ secrets.

For the same communication complexity, we show how to do detectable secret sharing for $\mathcal{O}(n^4)$ secrets or to do (standard) verifiable secret sharing for $\mathcal{O}(n^3)$ secrets. Looking ahead, we improve the basic scheme in the following aspects, each giving a factor of $\mathcal{O}(n^2)$ improvement for our detectable secret sharing:

(1) Packing: The bivariate polynomial $S(x, y)$ in the basic construction contains only a single secret, located at $S(0, 0)$. This is the best possible when sharing a bivariate polynomial of degree- t in both x and y : The t shares of the corrupted parties, together with the secret, fully determine the bivariate polynomial. In our detectable secret sharing scheme, the dealer shares a bivariate polynomial of degrees greater than t in *both* x and y . This allows planting $\mathcal{O}(n^2)$ secrets. The verification that all parties hold shares on the same bivariate polynomial is much more challenging because the degrees of all the univariate polynomials are greater than t . Nevertheless, we obtain binding with asymptotically the same cost as the basic scheme, therefore we already obtain an improvement of $\mathcal{O}(n^2)$ over the basic scheme.

Moreover, once the degree in both dimensions is greater than t , then reconstruction might

fail because the underlying codeword is of degree greater than t , and the parties cannot necessarily correct the errors if the adversary does not provide correct shares. Nevertheless, Reed-Solomon decoding guarantees that the honest parties can (efficiently) *identify* whether there is a unique decoding or not. We use this property to also *detect sufficiently many* corrupted parties. This suffices for constructing a detectable secret sharing scheme.

For (standard) verifiable secret sharing, we must make the degree in at least one of the dimensions to be at most t , to allow to always succeed in correcting errors. This allows us to pack “only” $\mathcal{O}(n)$ secrets and not $\mathcal{O}(n^2)$.

(2) Batching: The verification step of [29] requires broadcasting $\mathcal{O}(n^2)$ field elements by the dealer, and $\mathcal{O}(n)$ field elements by each party. Hence m independent instances (with the same dealer) require broadcasting of $\mathcal{O}(mn^2)$ field elements. First, we *balance* the protocol such that each party broadcasts at most $\mathcal{O}(n)$ field elements, including the dealer. Second, by designing a sharing protocol that is tailored for achieving cheap batching, *the broadcast cost for m independent instances remains the same as a single instance, i.e., it requires each party to broadcast $\mathcal{O}(n \log n)$ bits in all m executions combined.* By setting $m = \mathcal{O}(n^2)$ and implementing the broadcast over point-to-point, we get a detectable secret sharing of $\mathcal{O}(n^4)$ secrets (each is a field element of size $\mathcal{O}(\log n)$) at the cost of $\mathcal{O}(n^4 \log n)$ communication over the point-to-point channels. This is the second $\mathcal{O}(n^2)$ improvement over the basic scheme.

Our batched and packed detectable secret sharing protocol. For our discussion, assume that the dealer first chooses a polynomial $S(x, y)$ of degree $t + t/4$ in x and degree $t + t/4$ in y . We will use different parameters in the actual construction later,¹ but we choose $t + t/4$ for simplicity of exposition in this overview. Like the basic scheme, the view of the adversary consists of the pair of the univariate polynomials $S(x, i), S(i, x)$, for every $i \in I$, where $I \subseteq [n]$ is the set of indices of the corrupted parties (of cardinality at most t). This means that the adversary receives at most $2t(t + t/4 + 1) - t^2$ values, and therefore the dealer can still plant $(t/4 + 1)^2 \in \mathcal{O}(n^2)$ secrets in $S(x, y)$, which is fully determined by $(t + t/4 + 1)^2$ values. Concretely, it can plant for every $a \in \{0, \dots, t/4\}$ and $b \in \{0, \dots, t/4\}$ a secret at location $S(-a, -b)$.

Looking ahead, to allow *batching*, the dealer will choose m different bivariate polynomials $S_1(x, y), \dots, S_m(x, y)$, and all the parties will verify all the m instances simultaneously. To accept the shares, all instances must end up successfully. We follow the following two design principles:

1. *Broadcast is expensive; Each broadcast must be utilized in all m instances, not*

¹Our actual parameters are further optimized to pack more secrets.

just in one instance.

2. *Detection: Whenever a party is detected as an obstacle for achieving agreement (a foe), we should make it a “friend”, or more precisely, we neutralize its capacity to obstruct further, and utilize it to achieve agreement on a later stage.*

We focus on sharing of one instance for now, while keeping these design principles in mind. Along the way, we also discuss how to keep the broadcasts of the dealer low for all m instances simultaneously, and we will show how to reduce the broadcasts of other parties later on. We follow a similar structure to that of the basic scheme:

1. **Sharing:** The dealer sends $f_i(x), g_i(y)$ to each party P_i .
2. **Pairwise checks:** Each pair of parties exchange sub-shares. In case of a mismatch, a party broadcasts a complaint $\text{complaint}(i, j, f_i(j), g_i(j))$.

The dealer now has to resolve the complaints. In the basic protocol, when the dealer identifies party P_i as corrupted, the dealer simply broadcasts the “correct” $(S(x, i), S(i, y))$ so that everyone can verify that the shares are consistent. However, this leads to $\mathcal{O}(n^2)$ values being broadcasted, and $\mathcal{O}(mn^2)$ values in the batched case. Instead, in our protocol, the dealer just marks P_i as corrupted and adds it to a set $\text{CONFLICTS} \subset [n]$ which is initially empty. It broadcasts the set CONFLICTS . This set should be considered as “parties that had false complaints” from an honest dealer’s perspective. There are three cases to consider:

1. The dealer is discarded: This might happen, e.g., if two parties complained on each other and none of them is in CONFLICTS . In this case, it is clear that the dealer is corrupted, and all parties can just discard it.
2. If the dealer is not discarded and $|\text{CONFLICTS}| > t/4$, then we have **large conflict**. The dealer identified a large set of conflicts (note that if the dealer is honest, then CONFLICTS contains only corrupted parties). Instead of publicly announcing the polynomials $f_i(x), g_i(y)$ of the identified corrupted parties, the dealer simply restarts the protocol. In the new iteration, the shares of parties in CONFLICTS are publicly set to 0. That is, it chooses a new random bivariate polynomial $S(x, y)$ that hides the same secrets as before, this time under the additional constraints that $S(x, i) = S(i, y) = 0$ for every $i \in \text{CONFLICTS}$.

The dealer does not broadcast the shares of parties in CONFLICTS ; all the parties know that they are 0s. When each party receives its new pair of shares $f_j(x), g_j(y)$, it also verifies that $f_j(i) = g_j(i) = 0$, and if not, it raises a complaint. Parties in CONFLICTS cannot raise any complaints. Furthermore, observe that the outcome of “large conflict”

might occur only $\mathcal{O}(1)$ times; if the dealer tries to exclude more than t parties total, then the dealer is publicly discarded.

When batching over m instances, we choose the shares of the set CONFLICTS to be 0 in all instances. Thus, the dealer uses a broadcast of $\mathcal{O}(n \log n)$ bits, i.e., the set CONFLICTS, and by restarting the protocol it made the shares of parties in CONFLICTS public in all m executions. Thus, we get the same effect as broadcasting $m|\text{CONFLICTS}|$ pairs of polynomials (i.e., broadcasting $\mathcal{O}(m \cdot n^2 \log n)$ bits). This follows exactly our first design principle.

3. If $|\text{CONFLICTS}| \leq t/4$ then the dealer proceeds with the protocol. It has to reconstruct the f and g polynomials of all parties in CONFLICTS.

Before we proceed, let's highlight what guarantees we have so far: when the dealer is honest, then all the parties in CONFLICTS are corrupted. Moreover, if in some iteration there were more than $t/4$ identified conflicts by the dealer, those corrupted parties are eliminated, and they have shares that all parties know (i.e., 0) and are consistent with the shares of the honest parties. This turns a “foe” into a “friend”, as our second design principle.

When the dealer is corrupted, then all parties that are not in CONFLICTS have shares that define a unique bivariate polynomial, and we have binding. Specifically, if the shares of two honest parties do not agree with each other, then they both complain on each other, and the dealer must include one of them in CONFLICTS. Therefore, all honest parties that are not in CONFLICTS (assuming that the dealer was not publicly discarded) hold shares that are consistent with each other. Moreover, there is one more important property: Honest parties that were excluded in previous iterations (and now their shares are 0) also hold shares that are consistent with the honest parties that are not in CONFLICTS. In particular, if we indeed proceed, then there are at most $t/4$ honest parties who do not hold shares on the polynomial. This means that there are $2t + 1 - t/4$ honest parties that have shares on the bivariate polynomial – not only do we have binding, but we also have some redundancy! This redundancy will be crucial for our next step as we show below.

However, there might still be up to $t/4$ honest parties (in CONFLICTS) that do not have shares on the correct polynomial. The rest of the protocol is devoted to reconstructing their shares. We call this phase reconstruction of the shares of honest parties in CONFLICTS. However, before proceeding to the reconstruction, we first describe how to batch over m instances.

Batching Complaints. Consider sharing m instances simultaneously with the same dealer. In the above description, we already described how the dealer's broadcast is just the set

CONFLICTS, which require $\mathcal{O}(n \log n)$ bits, independent of m . However, the broadcast of other parties depends on m . Specifically:

1. A party P_i broadcasts $\text{complaint}(i, j, f_i(j), g_i(j))$ when it receives a wrong share from some party P_j .
2. A party P_i broadcast complaint if the share it received do not agree with the parties that are publicly 0. Recall that in that case, the dealer must include P_i in CONFLICTS.

It suffices to complain in only one of the instances, say the one with the lexicographically smallest index. This follows our first design principle. If two parties P_i and P_j do not agree in $\ell < m$ of the instances, they will both file a joint complaint with the same minimal index. Thus, we have a joint complaint, and in order to not be discarded, the dealer must include either i or j in CONFLICTS. Thus, we still have the guarantee that if two honest parties are not in CONFLICTS then their shares must be consistent, now in *all* m executions.

Likewise, if some party P_i receives from the dealer private shares where on points of some parties that were excluded it does not receive 0s, it essentially requires to be part of CONFLICTS. Thus, there is no need to make m requests, it suffices to make just one such request.

Reconstruction of the shares of honest parties in CONFLICTS. Going back to the last step of the sharing process, each party P_j in CONFLICTS wishes to reconstruct its pair of polynomials $(f_j(x), g_j(y))$. Towards that end, each party P_k that are not in CONFLICTS send to P_j , privately, the values $(f_j(k), g_j(k))$. P_j therefore is guaranteed to receive $2t + 1 - t/4$ correct points. However, the polynomials are of degree $t + t/4$, and we need $2t + t/4$ “correct values” to eliminate t errors. This means that if the adversary introduces more than $t/2$ incorrect values, P_j does not have unique decoding. If this is the case, then P_j broadcasts a complaint $\text{complaint}(j)$, insisting that its shares will be publicly reconstructed. As we will see, when batching over m executions, it is enough to make one public complaint in one execution, say the lexicographically smallest one, let’s denote it as $\beta \in [m]$. Resolving this instance will help to resolve all other m instances.

Upon receiving $\text{complaint}(j, \beta)$, each party P_k broadcasts $\text{reveal}(k, j, f_k(j), g_k(j))$ for the β th instance. Thus, we will have at least $2t + 1 - t/4$ correct values that are public. Moreover, corrupted parties might now reveal values that are different than what they have previously sent privately, and we might already have unique decoding. In any case, with each value that was broadcasted and is wrong, the dealer adds the identity of the party that broadcasted the wrong value into a set Bad . It then broadcasts the set Bad , and all parties can check that when excluding parties in Bad then all other values define a unique polynomial, and

all public points (excluding Bad) lie on this polynomial. Otherwise, the dealer is publicly discarded. Note that it is enough to broadcast one set Bad for all party $j \in \text{CONFLICTS}$ and for all m instances. If $|\text{Bad}| > t/2$, we restart the protocol, again giving shares 0 to parties in Bad (as long as the total number of parties that the dealer excluded does not exceed t).

At this point, if we did not restart and the dealer was not discarded, then it must be that P_j can reconstruct its polynomials $f_j(x), g_j(y)$ in all m instances. First, in the β th instance (that was publicly resolved), we know that we have $2t + 1 - t/4$ public points that are “correct” and that the dealer could have excluded at most $t/2$ parties. Therefore, there are more than $t + t/4 + 1$ correct points even if the dealer excludes up to $t/2$ honest parties (recall that it cannot exclude more than $t/2$). Those correct points uniquely determine a polynomial of degree $t + t/4$, and therefore, since all points after excluding parties in Bad lie on one unique polynomial, it must be that this polynomial is the correct one.

Using the information learned in the resolved instance party P_j can uniquely decode all other m instances. Specifically, there is no unique decoding in a particular instance only if P_j received more than $t/2$ wrong private shares. When going publicly, some parties might announce different values than what they first told P_j privately. P_j can compare between the polynomial reconstructed in the β th instance to the initial values it received privately from the parties, and identify all parties that sent it wrong shares. Denote this set as localBad_j . It must hold that this set contains more than $t/2$ corrupted parties. Now, in each one of the other instances, ignore all parties in localBad_j . This implies that the remaining values are of distance at most $t/2$ from a correct word, i.e., they contain at most $t/2$ errors. Moreover, it is guaranteed that honest parties are not eliminated, and we still have at least $2t + 1 - t/4$ correct points. Therefore, P_j guarantees to have unique decoding in all m instances.

Detectable and Robust Reconstruction. So far, we described the sharing procedure. While we do not use the reconstruction of detectable secret sharing directly (we will use private reconstruction, and parties never reconstruct all secrets), we briefly describe it for completeness. To reconstruct polynomials $S_1(x, y), \dots, S_m(x, y)$ that were shared with the same dealer, we follow a similar step as reconstruction towards parties in CONFLICTS, but with reconstructing all polynomials: Each party sends (privately) the f -shares, the parties try to privately reconstruct g_i -polynomials for all $i \in [n]$, and interpolate the bivariate polynomials from the g_i -polynomials. If some party does not succeed in uniquely reconstructing some g_i -polynomial, then it asks to go public. For each party P_j , it is enough to publicly reconstruct one g_i -polynomial that it did not succeed to reconstruct privately, and from that, P_j can reconstruct all other shares (by ignoring the new privately detected parties).

However, as before, the adversary can cause the reconstruction to fail. When it does so,

the dealer is guaranteed to detect more than $t/2$ corruptions. Moreover, if the dealer already detected at least $t/2$ corruptions during the sharing phase, then those parties cannot fail the reconstruction, and reconstruction is guaranteed. Note that the cost of the reconstruction is $\mathcal{O}(mn^2 \log n)$ over point-to-point channels, plus each party has to broadcast at most $\mathcal{O}(n \log n)$ bits, again, independent of m .

Reconstruction for VSS. Recall that for VSS, we set the degree of y in each bivariate polynomial to t . This implies that all parties can reconstruct all g -polynomials using Reed-Solomon error correction and we never have to resolve complaints publicly. Moreover, the adversary can never cause any failure. The cost is therefore $\mathcal{O}(mn^2 \log n)$ over point-to-point channels, and VSS robust reconstruction is always guaranteed.

We refer the reader to Section 4.4 for our packed secret sharing scheme for a single polynomial, and to Section 4.5 for the batched version.

4.2.2 Our MPC Protocol

Our MPC protocol follows the following structure: an offline phase in which the parties generate Beaver triplets [23], and an online phase in which the parties compute the circuit while consuming those triples.

Beaver triplets generation. Our goal is to distribute shares of random secret values a, b and c , such that $c = ab$. If the circuit contains C multiplication gates, then we need C such triplets. Towards that end, we follow the same steps as in [50], and generate such triplets in two stages:

1. **Triplets with a dealer:** Each party generates shares of a_i, b_i, c_i such that $c_i = a_i \cdot b_i$. We generate all the triplets in parallel using expected $\mathcal{O}(1)$ rounds. We will elaborate on this step below in Section 4.2.3. Our main contribution is in improving this step. In our protocol, each party acts as a dealer to generate mn triplets. This step requires an overall cost of $\mathcal{O}(n^4 \log n + mn^3 \log n)$ point-to-point communication for all the parties together. Later, these mn^2 triplets will be used for generating $\mathcal{O}(mn^2)$ triplets overall. Looking ahead, we will use $m = C/n^2$ and this step costs $\mathcal{O}(n^4 \log n + Cn \log n)$. Previously, the best known [50] used $\mathcal{O}(n^3 \log n)$ point-to-point and $\mathcal{O}(n^3 \log n)$ broadcast for generating just a single triplet for one dealer. That is, for $\mathcal{O}(mn^2)$ triplets this is $\mathcal{O}(mn^5 \log n)$ broadcast which costs at least $\Omega(mn^6 \log n)$ over point-to-point. We therefore improve in a factor of $\mathcal{O}(n^3)$.
2. **Triplets with no dealer:** Using triplet extraction of [50], we can extract from a total of C triplets with a dealer, $\mathcal{O}(C)$ triplets where no party knows the underlying values.

That is, if n parties generate C/n triplets each, then we have a total of C triplets and we can extract from it $\mathcal{O}(C)$ triplets. This step costs $\mathcal{O}(n^2 \log n + Cn \log n)$.

Putting it all together, for generating C triplets we pay a total of $\mathcal{O}(n^4 \log n + Cn \log n)$ and constant expected number of rounds.

The MPC protocol then follows the standard structure where each party shares its input, and the parties evaluate the circuit gate-by-gate, or more exactly, layer-by-layer. In each multiplication gate, the parties have to consume one multiplication triple. Using the method of [50], if the i th layer of the circuit contains C_i multiplications (for $i \in [D]$, where D is the depth of the circuit), the evaluation costs $\mathcal{O}(n^2 \log n + C_i \cdot n \log n)$. Summing over all layers, this is $\sum_{i \in [D]} (n^2 + nC_i) \log n = (Dn^2 + Cn) \log n$. Together with the generation of the triplets, we get the claimed $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$ cost as in Theorem 4.1.3. We refer the reader to Section 4.8 for further details on our MPC protocol.

4.2.3 Multiplication Triplets with a Dealer

As mentioned, a building block which we improve in a factor of $\mathcal{O}(n^3)$ over the state-of-the-art is multiplication triplets with a dealer. The goal is that given a dealer, to distribute shares of secret values $\vec{a}, \vec{b}, \vec{c}$ such that for every i it holds that $c_i = a_i b_i$. Towards this end, the dealer plants \vec{a} into some bivariate polynomial $A(x, y)$ using our verifiable secret sharing scheme. It plants \vec{b} into $B(x, y)$ and \vec{c} into $C(x, y)$ in a similar manner. Note that we use verifiable secret sharing here, since we want to output the triplets shared via degree- t polynomials (which is utilized by our MPC protocol). So we can plant only $\mathcal{O}(n)$ values in each one of them. Then, the dealer has to prove, using a distributed zero-knowledge protocol, that indeed $c_i = a_i b_i$ for every i . The zero-knowledge proof uses sharing and computations on the coefficients of the polynomials used for sharing $\vec{a}, \vec{b}, \vec{c}$, i.e., if we shared $\mathcal{O}(M)$ triplets, then the zero-knowledge involves sharing of $\mathcal{O}(Mn)$ values. However, since the dealer is involved in the sharing and the reconstruction of those values, we do not need full-fledged secret sharing scheme, and we can use the lighter detectable secret sharing. This scheme enables us to share $\mathcal{O}(Mn)$ values at the same cost of “strong” verifiable secret sharing of $\mathcal{O}(M)$ values.

In a more detail, after verifiable sharing A, B and C , the dealer needs to prove that for every $a \in \{0, \dots, t/4\}$ it holds that $C(-a, 0) = A(-a, 0) \cdot B(-a, 0)$. Towards that end, for every $a \in \{0, \dots, t/4\}$ it considers the polynomial

$$E_{-a}(y) = A(-a, y) \cdot B(-a, y) - C(-a, y) = e_{-a,0} + e_{-a,1}y + \dots + e_{-a,2t}y^{2t}$$

and its goal is to show that the degree- $2t$ polynomial $E_{-a}(y)$ evaluates to 0 on each $y \in$

$\{0, \dots, -t/4\}$. The dealer secret-shares all the coefficients $(e_{-a,k})$ for $a \in \{0, \dots, t/4\}$ and $k \in \{0, \dots, 2t\}$ using our detectable secret sharing scheme, by packing them into several bivariate polynomials $E(x, y)$. Note that there are $\mathcal{O}(n^2)$ coefficients to share, and each polynomial $E(x, y)$ can pack $(t/4 + 1)^2$ secrets.¹ Thus, we actually share a constant number (precisely 8) of polynomials to share all the coefficients.

Using linear combinations over the shares, the reconstruction protocol privately reconstructs towards P_j (for every $j \in [n]$) the evaluation of $E_{-a}(y)$ on j , i.e., $E_{-a}(j)$, for each $a \in \{0, \dots, t/4\}$. This is performed in a similar manner to the reconstruction of shares of honest parties in CONFLICTS in our detectable secret sharing protocol. Each P_j can then verify that $E_{-a}(j) = A(-a, j) \cdot B(-a, j) - C(-a, j)$, and if not, it can raise a public complaint. The parties can then open the shares of P_j on A, B, C publicly, and also the value $E_{-a}(j)$. If indeed $E_{-a}(j) \neq A(-a, j) \cdot B(-a, j) - C(-a, j)$, then the dealer is discarded.

Moreover, again using linear evaluations over the shares and reconstruction, the parties can obtain $E_{-a}(0)$ for every $a \in \{0, \dots, t/4\}$ and verify that it equals 0. If indeed $E_{-a}(j) = A(-a, j) \cdot B(-a, j) - C(-a, j)$ for $2t + 1$ such js , then $E_{-a}(y) = A(-a, y) \cdot B(-a, y) - C(-a, y)$ as those are two polynomials of degree $2t$ that agree on $2t + 1$ points. Moreover, if indeed $E_{-a}(0) = 0$ for every $a \in \{0, \dots, t/4\}$, then $C(-a, 0) = A(-a, 0) \cdot B(-a, 0)$ for every $a \in \{0, \dots, t/4\}$, as required.

The above description is a bit oversimplified. Recall that the coefficients of E are shared using only detectable secret sharing. This means that the private reconstruction towards some P_j might fail. In that case, P_j will ask to perform public reconstruction, and the adversary learns $E_{-a}(j)$ on a point $j \notin I$. This is a leakage because the reconstruction was meant to be private and becomes public. The good news is that the outcome of each such public reconstruction is that party P_j identifies at least $t/2$ corruptions in localBad_j , and all the later reconstructions towards it must succeed.

As a result, the adversary may learn up to $n - t$ reconstructions that it was not supposed to learn. Whenever this occurs, we cannot use the entire polynomials that are involved (which pack $\mathcal{O}(n)$ triplets). If a “pack” of triplets requires a public reconstruction, we discard the whole “pack”. On the positive side, this can happen at most once per party. Moreover, since the multiplication triplets are just random and do not involve secret inputs, we can just sacrifice them. This means that for generating m “packs” of triplets, we need to start with batching $\mathcal{O}(m + n)$ “packs” of triplets. This additional overhead does not affect the overall complexity, but it makes the functionalities and the protocol a bit more involved. We refer

¹Again, in the actual construction we will use different dimensions, but we keep using a bivariate polynomial with degree $t + t/4$ in both x and y for simplicity.

the reader to Sections 4.6 and 4.7 for further details.

Organization. The rest of this chapter is organized as follows. After some Preliminaries (Section 4.3) we focus on our packed (Section 4.4) and batched (Section 4.5) secret sharing. We then discuss our packed (Section 4.6) and batched (Section 4.7) multiplication triplets with a dealer, and conclude with the MPC protocol in Section 4.8.

4.3 Preliminaries

4.3.1 Network Model and Notations

We consider a synchronous network model where the parties in $\mathcal{P} = \{P_1, \dots, P_n\}$ are connected via pairwise private and authenticated channels. Additionally, for some of our protocols we assume the availability of a broadcast channel, which allows a party to send an identical message to all the parties. The detailed description of the network model and security definition is discussed in Section 2.

Our protocols are defined over a finite field \mathbb{F} where $|\mathbb{F}| > n + t/2 + 1$. We denote the elements by $\{-t/2, -t/2+1, \dots, 0, 1, \dots, n\}$. We use $\langle v \rangle$ to denote the degree- t Shamir-sharing of a value v among parties in \mathcal{P} .

4.3.2 Bivariate Polynomials and Secret Embedding

A degree (l, m) -bivariate polynomial over \mathbb{F} is of the form $S(x, y) = \sum_{i=0}^l \sum_{j=0}^m b_{ij} x^i y^j$ where $b_{ij} \in \mathbb{F}$. The polynomials $f_i(x) = S(x, i)$ and $g_i(y) = S(i, y)$ are called i th f and g univariate polynomials of $S(x, y)$ respectively. In our protocol, we use $(t+t/2, t+d)$ -bivariate polynomials where $d \leq t/4$, and the i th f and g univariate polynomials are associated with party P_i for every $P_i \in \mathcal{P}$.

We view a list of $(t/2 + 1)(d + 1)$ secrets SECRETS as a $(t/2 + 1) \times (d + 1)$ matrix. We then say that the set SECRETS is *embedded* in a bivariate polynomial $S(x, y)$ of degree $(t + t/2)$ in x and $(t + d)$ in y if for every $a \in \{0, \dots, t/2\}$ and $b \in \{0, \dots, d\}$ it holds that $S(-a, -b) = \text{SECRETS}(a, b)$.

4.3.3 Simultaneous Error Correction and Detection of Reed-Solomon Codes

We require the following coding-theory related results which are already discussed in Section 2 and we reiterate here with specific parameters used in our protocols. Let C be an Reed-Solomon (RS) code word of length N , corresponding to a k -degree polynomial (containing $k + 1$ coefficients). Assume that at most t errors can occur in C . Let \bar{C} be the word

after introducing error in C in at most t positions. Let the distance between C and \bar{C} be s where $s \leq t$. Then there exists an *efficient* decoding algorithm that takes \bar{C} and a pair of parameters (e, e') as input, such that $e + e' \leq t$ and $N - k - 1 \geq 2e + e'$ hold and gives one of the following as output:

1. Correction: output C if $s \leq e$, i.e. the distance between C and \bar{C} is at most e ;
2. Detection: output “more than e errors” otherwise.

Theorem 4.3.1 ([49, 87]). *Let C be an Reed-Solomon (RS) code word of length N , corresponding to a k -degree polynomial (containing $k + 1$ coefficients). Let \bar{C} be a word of length N such that the distance between C and \bar{C} is at most t . Then RS decoding can correct up to e errors in \bar{C} to reconstruct C and detect the presence of up to $e + e'$ errors in \bar{C} if and only if $N - k - 1 \geq 2e + e'$ and $e + e' \leq t$.*

A couple of corollaries follows from the above theorem that we will use in our work.

Corollary 4.3.2. *Let C and \bar{C} be as in Theorem 4.3.1 with $N = 2t + 1 + d + t/2$ and $k = t + d$ for any $d > 0$.*

1. *Then RS decoding can correct up to $t/2$ errors in \bar{C} , or detect the presence of up to t errors in \bar{C} .*
2. *If $t' > t/2$ errors are known in code word \bar{C} , then the remaining $t - t'$ errors in \bar{C} can be corrected from the truncated code word C' obtained by removing the t' error points from \bar{C} .*

Proof. The first item follows because $N - k - 1 = 2t + 1 + d + t/2 - t - d - 1 = t + t/2$, $2e + e' = t + t/2$ and $e + e' = t$ hold. The second item holds because $(N - t') - k - 1 = 2t + 1 + d + t/2 - t' - t - d - 1 = t + t/2 - t'$. And so $N - t' - k - 1 = t + t/2 - t' \geq 2(t - t')$ holds true for all $t' > t/2$. \square

Corollary 4.3.3. *Let C and \bar{C} be as in Theorem 4.3.1 with $N = 3t + 1$ and $k = t + t/2$.*

1. *Then RS decoding can correct up to $t/2$ errors and detect the presence of up to t errors in \bar{C} .*
2. *If $t' > t/2$ errors are known in \bar{C} , then $t - t'$ errors can be corrected from the truncated codeword C' obtained from \bar{C} after removing the t' error points.*

Proof. The first item follows since $N - k - 1 = t + t/2$, $2e + e' = t + t/2$ and $e + e' = t$ hold. The second item follows since $(N - t') - k - 1 = 3t + 1 - t' - t - t/2 - 1 = t + t/2 - t'$. And so $N - t' - k - 1 = t + t/2 - t' \geq 2(t - t')$ holds true for all $t' > t/2$. \square

Corollary 4.3.4. *Let C and \bar{C} be as in Theorem 4.3.1 with $N = 3t + 1$ and $k = t + d$ where $d \leq t/4$.*

1. *Then RS decoding can correct up to $t/2$ errors and detect the presence of up to t errors in \bar{C} .*
2. *If $t' > t/2$ errors are known in \bar{C} , then $t - t'$ errors can be corrected from the truncated codeword C' obtained from \bar{C} after removing the t' error points.*

Proof. The first item follows since $N - k - 1 = 2t - d \geq t + 3t/4$, $2e + e' = t + t/2$ and $e + e' = t$ hold. The second item follows since $(N - t') - k - 1 = 3t + 1 - t' - t - d - 1 = 2t - d - t' \geq t + 3t/4 - t'$. And so $N - t' - k - 1 \geq t + 3t/4 - t' \geq 2(t - t')$ holds true for all $t' > t/2$. \square

4.3.4 Parallel Broadcast

In our MPC, we use parallel broadcast that relates to the case where n parties wish to broadcast a message of size L bits in parallel, as captured in the following functionality.

Functionality 4.3.5: $\mathcal{F}_{\text{BC}}^{\text{parallel}}$

The functionality is parameterized with a parameter L .

1. Each $P_i \in \mathcal{P}$ sends the functionality its message $M_i \in \{0, 1\}^L$.
 2. The functionality sends to all parties the message $\{M_i\}_{i \in [n]}$.
-

The work of [7] presents an instantiation with the following security and complexity. Also note that, when some party has smaller message than L bits, it can pad with default values to make an L bit message.

Theorem 4.3.6 ([7]). *There exists a perfectly-secure parallel broadcast with optimal resilience of $t < n/3$, which allows n parties to broadcast messages of size L bits each, at the cost of $\mathcal{O}(n^2L)$ bits communication, plus $\mathcal{O}(n^4 \log n)$ expected communicating bits. The protocols runs in constant expected number of rounds.*

4.4 Packed Secret Sharing

In this section we present our secret sharing scheme. In the introduction, we mentioned that we have two variants: regular verifiable secret sharing, and a novel detectable secret sharing. The protocol presented in this section fits the two primitives, where the difference is obtained by using different parameters in the bivariate polynomial, as we will see shortly. In this section, we still do not “batch” over multiple polynomials; the dealer share just a single polynomial. In Section 4.5 we provide details on the batched version. The packed secret sharing protocol consists of the following building blocks:

1. The dealer chooses a bivariate polynomial $S(x, y)$ of degree $3t/2$ in x and degree $t + d$ in y , where its secret are embedded in S . We should think of d as 0 or $t/4$. Unlike presented in Section 4.2.1, we have two different parameters for x and y . Looking ahead, for verifiable secret sharing, we use $d = 0$. For detectable secret sharing, we can use $d \in [1, t/4]$ (packing $\mathcal{O}((d + 1)n)$ secrets).
2. The dealer tries to share $S(x, y)$ using a functionality called $\mathcal{F}_{\text{ShareAttempt}}$ (see Functionality 4.4.1). At the end of this functionality, the sharing attempt might have the following three outcomes: (a) discard – the dealer is discarded; (b) (detect, CONFLICTS) - a large set of conflicts was detected and the protocol will be restarted; (c) proceed, in which case all parties also receive a set CONFLICTS (of size at most $t/2 - d$) of parties that still did not receive shares. All honest parties not in CONFLICTS hold shares that define unique bivariate polynomial of the appropriate degree. See Section 4.4.1 for further details.
3. The goal is now to let parties in CONFLICTS to learn their shares. Since the degrees of the bivariate polynomial is not symmetric, we first reconstruct the g -share (of degree $t + d < 3t/2$), and then the f -share (of degree $3t/2$). Reconstruction of g -polynomial is described in Section 4.4.2. The reconstruction of f -polynomial is similar, and is discussed in Section 4.4.3.

We first present the different building blocks, and then in Section 4.4.4 we provide the protocol (and functionality) for packed secret sharing, that uses those building blocks.

4.4.1 Sharing Attempt

We start with the description of the functionality.

Functionality 4.4.1: Sharing Attempt– $\mathcal{F}_{\text{ShareAttempt}}$

The functionality is parameterized with the set of corrupted parties $I \subset [n]$.

1. All the honest parties send to $\mathcal{F}_{\text{ShareAttempt}}$ a set ZEROS $\subset [n]$. For an honest dealer, it holds that ZEROS $\subseteq I$. $\mathcal{F}_{\text{ShareAttempt}}$ sends the set ZEROS to the adversary.
2. The dealer sends a polynomial $S(x, y)$ to $\mathcal{F}_{\text{ShareAttempt}}$. When either the polynomial is not of degree at most $3t/2$ in x and at most $t + d$ in y , or for some $i \in \text{ZEROS}$ it holds that $S(x, i) \neq 0$ or $S(i, y) \neq 0$, $\mathcal{F}_{\text{ShareAttempt}}$ executes Step 4c to discard the dealer.
3. For every $i \in I$, $\mathcal{F}_{\text{ShareAttempt}}$ sends $(S(x, i), S(i, x))$ to the adversary. It receives back a set CONFLICTS such that CONFLICTS $\cap \text{ZEROS} = \emptyset$.¹ If the dealer is honest, then

¹To ease understanding and notion, we sometimes expect to receive from the adversary some sets or

$\text{CONFLICTS} \cup \text{ZEROS} \subseteq I$. If $|\text{CONFLICTS} \cup \text{ZEROS}| > t$ for a corrupt dealer, then $\mathcal{F}_{\text{ShareAttempt}}$ executes Step 4c to discard the dealer.

4. Output:

- (a) Detect: If $|\text{CONFLICTS}| > t/2 - d$, then send $(\text{detect}, \text{CONFLICTS})$ to all parties.
- (b) Proceed: Otherwise, send $(\text{proceed}, S(x, i), S(i, y), \text{CONFLICTS})$ for every $i \notin \text{CONFLICTS}$ and $(\text{proceed}, \perp, \perp, \text{CONFLICTS})$ to every $i \in \text{CONFLICTS}$.
- (c) Discard: send discard to all parties.

Protocol 4.4.2: Sharing Attempt– $\Pi_{\text{ShareAttempt}}$

Common input: The description of a field \mathbb{F} , parameter $d < t$.

Input: All parties input $\text{ZEROS} \subset [n]$. The dealer inputs a polynomial $S(x, y)$ with degree $3t/2$ in x and $t + d$ in y , such that for every $i \in \text{ZEROS}$ it holds that $S(x, i) = 0$ and $S(i, y) = 0$.

The protocol:

1. **(Dealing shares):** The dealer sends $(f_i(x), g_i(y)) = (S(x, i), S(i, y))$ to P_i for $i \notin \text{ZEROS}$. Each P_i for $i \in \text{ZEROS}$ sets $(f_i(x), g_i(y)) = (0, 0)$.
2. **(Pairwise Consistency Checks):**
 - (a) Each $i \notin \text{ZEROS}$ sends $(f_i(j), g_i(j))$ to every $j \notin \text{ZEROS}$. Let (f_{ji}, g_{ji}) be the values received by P_i from P_j .
 - (b) Each $i \notin \text{ZEROS}$ broadcasts $\text{complaint}(i, j, f_i(j), g_i(j))$ if (a) $f_{ji} \neq g_i(j)$ or $g_{ji} \neq f_i(j)$ for any $j \notin \text{ZEROS}$. For $j \in \text{ZEROS}$, P_i broadcasts $\text{complaint}(i, j, f_i(j), g_i(j))$ if $f_i(j) \neq 0$ or $g_i(j) \neq 0$.
3. **(Conflict Resolution):**
 - (a) The dealer sets $\text{CONFLICTS} = \emptyset$. For each $\text{complaint}(i, j, u, v)$ such that $u \neq S(j, i)$ or $v \neq S(i, j)$, the dealer adds i to CONFLICTS . The dealer broadcasts CONFLICTS .
 - (b) Discard the dealer if any one of the following does not hold: (i) $|\text{ZEROS} \cap \text{CONFLICTS}| = \emptyset$; (ii) $|\text{CONFLICTS} \cup \text{ZEROS}| \leq t$ (iii) if some P_i broadcasted $\text{complaint}(i, j, u_i, v_i)$ and P_j broadcasted $\text{complaint}(j, i, u_j, v_j)$ with $u_i \neq v_j$ or $v_i \neq u_j$, then CONFLICTS should contain either i or j (or both); (iv) if some P_i broadcasted $\text{complaint}(i, j, u, v)$ with $j \in \text{ZEROS}$ and $u \neq 0$ or $v \neq 0$, then $i \in \text{CONFLICTS}$.

inputs that satisfy some conditions. We do not necessarily verify the conditions in the functionality, and this is without loss of generality. For instance, in this step we require that the adversary sends a set CONFLICTS such that $\text{CONFLICTS} \cap \text{ZEROS} = \emptyset$. Instead, we can enforce that this is the case by resetting: $\text{CONFLICTS} = \text{CONFLICTS} \setminus \text{ZEROS}$.

4. **(Output):** Each P_i outputs discard when the dealer is discarded and (detect, CONFLICTS) when $|\text{CONFLICTS}| > t/2 - d$. Else, it outputs (proceed, \perp, \perp , CONFLICTS) when $i \in \text{CONFLICTS}$, and (proceed, $f_i(x), g_i(y)$, CONFLICTS) otherwise.
-

Lemma 4.4.3. Protocol 4.4.2, $\Pi_{\text{ShareAttempt}}$, perfectly-securely computes Functionality 4.4.1, $\mathcal{F}_{\text{ShareAttempt}}$, in the presence of a malicious adversary, controlling at most $t < n/3$.

Proof. The efficiency of the protocol can be verified by inspection. As for security, we prove the statement separately for the case of an honest dealer and of a corrupted dealer.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z . Initialize $\text{CONFLICTS} = \emptyset$.
2. Receive from the functionality the set ZEROS and a set of polynomials $(f_i(x), g_i(y))_{i \in I}$. For every $i \in I \setminus \text{ZEROS}$, send \mathcal{A} the pair $(f_i(x), g_i(y))$ as coming from the dealer to P_i .
3. For every $j \notin I$, and $i \notin \text{ZEROS}$, simulate P_j privately sending to P_i the pair $(f_j(i), g_j(i)) = (g_i(j), f_i(j))$.
4. Receive from the adversary values $(f_{i,j}, g_{i,j})$ for every $i \in I \setminus \text{ZEROS}$ and $j \notin I$. If $f_{i,j} \neq f_i(j)$ or $g_{i,j} \neq g_i(j)$, then simulate P_j broadcasting $\text{complaint}(j, i, g_i(j), f_i(j))$.
5. If the adversary broadcasts $\text{complaint}(i, j, u_i, v_i)$ with $u_i \neq f_i(j)$ or $v_i \neq g_i(j)$, then add i to CONFLICTS.
6. Simulate the dealer broadcasting CONFLICTS, and send CONFLICTS to the functionality.
7. Receive from the functionality the output. If (detect, CONFLICTS) received, then send it to the adversary. Otherwise, for each $i \in I \setminus \text{CONFLICTS}$, send (proceed, $f_i(x), g_i(y)$, CONFLICTS) and send (proceed, \perp, \perp , CONFLICTS) to each $i \in \text{CONFLICTS}$.

It is clear that since the protocol as well as the simulation is deterministic, the adversary's view in the real execution and ideal execution are identical. It thus remains to be shown that the output of the honest parties is the same in both these executions.

In the ideal execution, all the honest parties including the dealer hold the same set ZEROS with which they invoke the functionality. Moreover, an honest dealer always sends a valid $(3t/2, t + d)$ -bivariate polynomial $S(x, y)$ such that $S(x, i) = 0$ and $S(i, y) = 0$ holds for each $i \in \text{ZEROS}$. Thus, it is guaranteed that the honest parties never output discard. Consequently, each honest party P_i either outputs (detect, CONFLICTS) or

(proceed, $S(x, i), S(i, y), \text{CONFLICTS}$). The latter holds since no honest party belongs to CONFLICTS in the case of an honest dealer.

In the real execution, since the dealer is honest, it always holds a valid $(3t/2, t + d)$ -bivariate polynomial $S(x, y)$ such that $S(x, i) = 0$ and $S(i, y) = 0$ holds for each $i \in \text{ZEROS}$. Moreover an honest dealer is never in conflict with another honest party and hence CONFLICTS may consist of only the corrupted parties. We first show that an honest dealer is never discarded in a real execution. To that end, observe that a dealer is discarded if and only if the following conditions hold:

1. $|\text{ZEROS} \cap \text{CONFLICTS}| \neq \emptyset$.
2. $|\text{CONFLICTS} \cup \text{ZEROS}| > t$.
3. If some P_i broadcasted complaint(i, j, u_i, v_i) and P_j broadcasted complaint(j, i, u_j, v_j) with $u_i \neq v_j$ or $v_i \neq u_j$ and $i, j \notin \text{CONFLICTS}$.
4. If some P_i broadcasted complaint(i, j, u, v) with $j \in \text{ZEROS}$ and $u \neq 0$ or $v \neq 0$ and $i \notin \text{CONFLICTS}$.

It is clear than none of the above conditions hold in the case of an honest dealer, and hence the honest parties do not output discard. We thus have the following two cases to consider:

1. **There exists an honest party which outputs (detect, CONFLICTS) in the real execution:** In such a case, we claim that all the honest parties output (detect, CONFLICTS). Note that an honest party outputs (detect, CONFLICTS) if and only if $|\text{CONFLICTS}| > t/2 - d$. The set CONFLICTS is broadcasted by the dealer, hence all the honest parties output (detect, CONFLICTS). Since the simulator emulates the interaction of the honest parties with the adversary as in the real execution of the protocol, all the simulated honest parties hold the same set CONFLICTS as the honest parties in the real execution. In this case, the simulator invokes the functionality with this set, which in turn sends (detect, CONFLICTS) to all the honest parties in the ideal execution, which is identical to the output of the honest parties in the real world.
2. **No honest party outputs (detect, CONFLICTS) in the real execution:** In this case, we show that each honest party outputs (proceed, $S(x, i), S(i, y), \text{CONFLICTS}$). Observe that since the set CONFLICTS is broadcasted by the dealer and no honest party outputs detect, it must hold that $|\text{CONFLICTS}| \leq t/2 - d$. Further, the polynomials $S(x, i), S(i, y)$ held by a party P_i are sent by the dealer and do not change during the protocol execution. Moreover, as mentioned, an honest party never belongs to CONFLICTS .

Hence, each honest party outputs the polynomials consistent with the dealer's polynomial $S(x, y)$. That is, each honest P_i outputs (proceed, $S(x, i)$, $S(i, y)$, CONFLICTS). As before, since the simulator emulates the interaction of the honest parties with the adversary as in the real execution of the protocol, all the simulated honest parties hold the same set CONFLICTS as the honest parties in the real execution. The simulator invokes the functionality with this set, causing all the honest parties P_i to output (proceed, $S(x, i)$, $S(i, y)$, CONFLICTS) in the ideal execution. This is identical to the output of the honest parties in the real world.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z .
2. Receive from the functionality the set ZEROS. Simulate running the protocol on behalf of the honest parties with ZEROS as input.
3. There are three cases to consider:
 - (a) Discard: If the output of some simulated honest party P_j is discard, then send $S(x, y) = y^{t+d+1}$ to the functionality together with CONFLICTS = \emptyset (in that case, $S(x, y)$ is being rejected by the functionality and all honest parties output discard).
 - (b) Otherwise, if some honest party output (detect, CONFLICTS) then it must hold that $|\text{CONFLICTS}| > t/2 - d$. Send $(S(x, y) = y^{t+d}, \text{CONFLICTS})$ to the functionality. In that case, the functionality would output (detect, CONFLICTS) to all parties.
 - (c) Otherwise, let J be an arbitrary set of $t + d + 1$ honest parties that are not in CONFLICTS. Note that since $|\text{CONFLICTS}| \leq t/2 - d$ we have at least $n - t/2 + d \geq 2t + 1 + t/2 + d$ parties that are not in CONFLICTS and therefore at least $3t/2 + d + 1$ honest parties not in CONFLICTS. Find the unique bivariate polynomial $S(x, y)$ in degree $3t/2$ in x and $t + d$ in y such that (a) $S(x, j) = f_j(x)$ for every $j \in J$. Send $S(x, y)$ together with CONFLICTS to the functionality.

Since the simulator emulates the honest parties as in the real execution of the protocol, the view of the adversary in the real and ideal execution is identical. It thus remains to show that the output of the honest parties in the real world and ideal world is the same. There are three cases to consider.

1. **There exists an honest party that outputs discard in the real world:** An honest party outputs discard only if verification fails at Step 3b. In this case, all the corresponding

messages are broadcasted and hence all the honest parties output \perp . Since the real and simulated executions are identical, all the simulated honest parties also output `discard`. Thus the simulator invokes the functionality as in Step 3a of the simulation, causing all the honest parties to receive `discard` in the ideal world.

2. **There exists an honest party that outputs $(\text{detect}, \text{CONFLICTS})$ in the real world:** This implies that for the set `CONFLICTS` broadcasted by the dealer, it holds that $|\text{CONFLICTS}| > t/2 - d$. Thus, each honest party outputs $(\text{detect}, \text{CONFLICTS})$ in the real world. Given that the simulated and real executions are identical, the simulated honest parties observe the same set `CONFLICTS`. The simulator in this case invokes the functionality as in Step 3b of the simulation, causing the honest parties to output $(\text{detect}, \text{CONFLICTS})$ in the ideal world.
3. **No honest party outputs `discard` or $(\text{detect}, \text{CONFLICTS})$ in the real world:** In this case, it means that $|\text{CONFLICTS}| \leq t/2 - d$ and parties output `proceed` in the real execution. We want to show that all the honest parties $j \notin \text{CONFLICTS}$ hold $f_i(x)$ and $g_i(y)$ consistent with a unique $(3t/2, t + d)$ -bivariate polynomial $S(x, y)$. Towards that end, observe that since $|\text{CONFLICTS}| \leq t/2 - d$ holds, we have that at least $3t/2 + d + 1$ honest parties are not in `CONFLICTS`. Let J be an arbitrary set of $t + d + 1$ honest parties that are not in `CONFLICTS`, and reconstruct the $(3t/2, t + d)$ -bivariate polynomial $S(x, y)$ from the set of degree- $(3t/2)$ univariate polynomials $(f_j(x))_{j \in J}$. We claim that the polynomials of all honest parties, not belonging to `CONFLICTS` lie on that polynomial.

To show that, we first claim that the g polynomial of each honest party P_j where $j \notin \text{CONFLICTS}$ agrees with S . Specifically, we have two cases here:

- For each honest party $j \in \text{ZEROS}$, it holds that $g_j(y) = S(j, y) = 0$. Specifically, for every $k \in J \cap \text{ZEROS}$ it trivially holds that $g_j(k) = f_k(j) = 0$. And for every $k \in J \setminus \text{ZEROS}$, it must hold that $g_j(k) = f_k(j) = 0$, as otherwise P_k would have raised a complaint and if the dealer does not include it in `CONFLICTS` then it would have been discarded. Since $g_j(y)$ and $S(j, y)$ are both degree- $(t + d)$ polynomials which agree in $t + d + 1$ points, $g_j(y) = S(j, y) = 0$ holds.
- For each honest party $j \notin \text{ZEROS}$, it holds that $g_j(y) = S(j, y)$. Specifically, for every $k \in J \cap \text{ZEROS}$ it must hold that $g_j(k) = f_k(j)$, as otherwise P_j would have raised a complaint and thus $j \in \text{CONFLICTS}$, which is a contradiction. Similarly, for every $k \in J \setminus \text{ZEROS}$ it must hold that $g_j(k) = f_k(j)$, as otherwise P_j and P_k would have raised a joint complaint and thus either $j \in \text{CONFLICTS}$ or $k \in \text{CONFLICTS}$, which is a contradiction. Finally, since $g_j(y)$ and $S(j, y)$ are both

degree- $(t + d)$ polynomials which agree in $t + d + 1$ points, $g_j(y) = S(j, y)$ holds. Thus, for each honest $j \notin \text{CONFLICTS}$, it holds that $g_j(y)$ is consistent with $S(x, y)$. Note that since $|\text{CONFLICTS}| \leq t/2 - d$, we have that the g polynomial of at least $3t/2 + d + 1$ honest parties is consistent with S . We now proceed to show that the f polynomial of each honest party P_j where $j \notin \text{CONFLICTS}$ agrees with S . For this, consider an arbitrary set K of $3t/2 + 1$ honest parties that are not in CONFLICTS . Then we have the following,

- For each honest party $j \in \text{ZEROS}$, it holds that $f_j(x) = S(x, j) = 0$. Specifically, for every $k \in K \cap \text{ZEROS}$ it trivially holds that $f_j(k) = g_k(j) = 0$. And for every $k \in K \setminus \text{ZEROS}$, it must hold that $f_j(k) = g_k(j) = 0$, as otherwise P_k would have raised a complaint and thus $k \in \text{CONFLICTS}$, which is a contradiction. Since $f_j(x)$ and $S(x, j)$ are both degree- $(3t/2)$ polynomials which agree in $3t/2 + 1$ points, $f_j(x) = S(x, j) = 0$ holds.
- For each honest party $j \notin \text{ZEROS} \cup \text{CONFLICTS}$, it holds that $f_j(x) = S(x, j)$. Specifically, for every $k \in K \cap \text{ZEROS}$ it must hold that $f_j(k) = g_k(j)$, as otherwise P_j would have raised a complaint and thus $j \in \text{CONFLICTS}$, which is a contradiction. Similarly, for every $k \in K \setminus \text{ZEROS}$ it must hold that $f_j(k) = g_k(j)$, as otherwise P_j and P_k would have raised a joint complaint and thus either $j \in \text{CONFLICTS}$ or $k \in \text{CONFLICTS}$, which is a contradiction. Finally, since $f_j(x)$ and $S(x, j)$ are both degree- $(3t/2)$ polynomials which agree in $3t/2 + 1$ points, it holds that $f_j(x) = S(x, j)$.

We conclude that if the honest parties output proceed, then each honest $j \notin \text{CONFLICTS}$ holds $f_j(x)$ and $g_j(y)$ consistent with a unique bivariate polynomial $S(x, y)$. Moreover, since the set CONFLICTS is broadcasted and it holds that $|\text{CONFLICTS}| \leq t/2 - d$, each honest P_j with $j \in \text{CONFLICTS}$ outputs $(\text{proceed}, \perp, \perp, \text{CONFLICTS})$. Since the simulated and real executions are identical, output of the simulated honest parties is the same as the honest parties in the real execution. Thus, in this case, the simulator invokes the functionality as in Step 3c of the simulation, which in turn ensures that the output of the honest parties is identical in the real and ideal executions.

□

4.4.2 Reconstruction of g -Polynomials in CONFLICTS

When invoking this functionality, we are guaranteed that the shares of the honest parties define a unique bivariate polynomial, and that the number of parties that are not in CONFLICTS

is at least $(n - t/2) + d$. The goal of this step is to reconstruct the g -polynomials for the parties in CONFLICTS, while the possible outcomes are: (i) the dealer is discarded; (ii) the dealer detects additional $t/2$ parties that it will make ZEROS in the next iteration; (iii) the protocol succeeds and all honest parties hold $g_j(y)$ as output.

Functionality 4.4.4: Reconstruction of g -Polynomials – $\mathcal{F}_{\text{rec-g}}$

1. **Input:**¹ All honest parties send to the functionality $\mathcal{F}_{\text{rec-g}}$ the sets $\text{ZEROS} \subset [n]$ and $\text{CONFLICTS} \subset [n]$, each honest $j \notin \text{CONFLICTS}$ sends $(f_i(x), g_i(y))$. Let $S(x, y)$ be the unique bivariate polynomial of degree at most $3t/2$ in x and at most $t + d$ in y that satisfies $f_j(x) = S(x, j)$ and $g_j(y) = S(j, y)$ for every $j \notin \text{CONFLICTS}$. Moreover, it holds that $n - |\text{CONFLICTS}| \geq 2t + 1 + t/2 + d$.
 2. $\mathcal{F}_{\text{rec-g}}$ sends $(\text{ZEROS}, \text{CONFLICTS}, (S(x, i), S(i, y))_{i \in I})$ to the adversary. If the dealer is corrupted, then $\mathcal{F}_{\text{rec-g}}$ sends $S(x, y)$ as well.
 3. It receives back from the adversary a message M .
 4. **Output:**
 - (a) If $M = \text{discard}$ and the dealer is corrupted, then $\mathcal{F}_{\text{rec-g}}$ sends discard to all parties.
 - (b) If $M = (\text{detect}, \text{Bad})$ with $\text{Bad} \cap (\text{ZEROS} \cup \text{CONFLICTS}) = \emptyset$ and $|\text{Bad}| > t/2$, and with $\text{Bad} \subseteq I$ in the case of an honest dealer, then $\mathcal{F}_{\text{rec-g}}$ sends $(\text{detect}, \text{Bad})$ to all parties.
 - (c) If $M = \text{proceed}$, then $\mathcal{F}_{\text{rec-g}}$ sends:
 - for each $j \in \text{CONFLICTS}$ the output $(\text{proceed}, \perp, S(j, y))$, and
 - for each $j \notin \text{CONFLICTS}$ send $(\text{proceed}, S(x, j), S(j, y))$.
-

Protocol 4.4.5: Reconstruct g -Polynomials in CONFLICTS – $\Pi_{\text{rec-g}}$

Input: All parties hold the same set CONFLICTS and ZEROS. Each honest party not in CONFLICTS holds a pair of polynomials $(f_i(x), g_i(y))$, and it is guaranteed that all the shares of honest parties lie on the same bivariate polynomial $S(x, y)$ with degree at most $3t/2$ in x and $t + d$ in y .

¹If not all honest parties send shares that lie on the same bivariate polynomial, or not all send inputs that satisfy the input assumptions as described, then no security is guaranteed. This can be formalized as follows. If the input assumptions do not hold, then the functionality sends to the adversary all the inputs of all honest parties, and lets the adversary to singlehandedly determine all outputs of all honest parties. This makes the protocol vacuously secure (since anything can be simulated).

The protocol:

1. Every party sets HAVE-SHARES = $[n] \setminus (\text{ZEROS} \cup \text{CONFLICTS})$.
 2. For every $j \in \text{CONFLICTS}$:
 - (a) Each party P_i for $i \in \text{HAVE-SHARES}$ sends $(i, f_i(j))$ to P_j .
 - (b) Let (i, u_i) be the value P_j received from P_i . Moreover, for every $i \in \text{ZEROS}$, consider (i, u_i) with $u_i = 0$. Given all $(i, u_i)_{i \notin \text{CONFLICTS}}$, P_j looks for a codeword of a polynomial of degree $t + d$ with a distance of at most $t/2$ from all the values it received (see Corollary 4.3.2, item 1). If there is such codeword, set $g_j(y)$ to be the unique Reed-Solomon reconstruction. If there is no such a unique codeword, then P_j broadcasts $\text{complaint}(j)$ and every party P_i for $i \in \text{HAVE-SHARES}$ broadcasts $\text{reveal}(i, j, f_i(j))$.
 3. The dealer sets $\text{Bad} = \emptyset$. For each $\text{reveal}(i, j, u)$ message broadcasted, the dealer verifies that $u = f_i(j)$. If not, then it adds i to Bad . The dealer broadcasts Bad .
 4. The parties go to Step 6a if one of the following is not true: (i) $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| \leq t$; (ii) $\text{Bad} \subset \text{HAVE-SHARES}$. The parties go to Step 6b if $|\text{Bad}| > t/2$.
 5. Otherwise, for every $j \in \text{CONFLICTS}$, if $\text{complaint}(j)$ was broadcasted, then the parties consider all the points $R_j = \{(i, u_i)\}$ such that $\text{reveal}(i, j, u_i)$ was broadcasted in Step 2b, and $i \in \text{HAVE-SHARES} \setminus \text{Bad}$, or $u_i = 0$ if $i \in \text{ZEROS}$. They verify if R_j defines a unique polynomial of degree $t + d$. If not, they go to Step 6a. Otherwise, P_j sets $g_j(y)$ to be that unique polynomial.
 6. **Output:**
 - (a) Discard the dealer: Output discard .
 - (b) Detect: Output $(\text{detect}, \text{Bad})$.
 - (c) Proceed: Each party $j \in \text{CONFLICTS}$ outputs $(\text{proceed}, \perp, g_j(y))$. All other parties P_j with $j \notin \text{CONFLICTS}$ output $(\text{proceed}, f_j(x), g_j(y))$.
-

Lemma 4.4.6. Protocol 4.4.5, $\Pi_{\text{rec-g}}$, perfectly securely computes Functionality 4.4.4, $\mathcal{F}_{\text{rec-g}}$, in the presence of a malicious adversary, controlling at most $t < n/3$. The protocol requires the transmission of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels, and each party broadcasts at most $\mathcal{O}(n \log n)$ bits.

Proof. We prove the statement separately for the case of an honest dealer and of a corrupted dealer.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z . Set $\text{HAVE-SHARES} = [n] \setminus (\text{ZEROS} \cup \text{CONFLICTS})$.
2. Receive from the functionality the sets $\text{ZEROS}, \text{CONFLICTS}$ and the polynomials $S(x, i), S(i, y)$ for every $i \in I$.
3. Set $\text{Bad} = \emptyset$. For every $i \in \text{CONFLICTS}$: (note that since the dealer is honest, then $I \subseteq \text{CONFLICTS}$. Thus, any such element is also in I)
 - (a) Simulate party $P_j, j \in \text{HAVE-SHARES}$ sending $(j, i, f_j(i)) = (j, i, g_i(j))$ to the adversary.
 - (b) If P_i broadcasts $\text{complaint}(i)$ then simulate party $P_j, j \in \text{HAVE-SHARES}$ broadcasting $\text{reveal}(j, i, g_i(j))$.
 - (c) Listen to all broadcasts $\text{reveal}(i', i, u_i)$ that the adversary sends for some $i' \in \text{HAVE-SHARES} \cap I$. If $u_i \neq g_i(i')$, then add i' to Bad .
4. Simulate the dealer broadcasting Bad . If $|\text{Bad}| > t/2$, then send $(\text{detect}, \text{Bad})$ to the functionality and halt.
5. Otherwise, send proceed to the functionality, and halt.

When the dealer is honest, we have that $\text{CONFLICTS} \subseteq I$. Moreover, the protocol is deterministic, and so is the simulator. By inspection, the view of the adversary in the real and ideal executions is identical. We now show that the output of honest parties is identical in the real and ideal world.

In the real world, first note that an honest dealer is discarded if and only if one of the following conditions holds:

1. $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| > t$.
2. $\text{Bad} \not\subseteq \text{HAVE-SHARES}$.
3. R_j does not define a unique polynomial of degree- $(t + d)$.

Note that for an honest dealer, an honest party never belongs to Bad and we have that $\text{ZEROS} \subseteq I$ and $\text{CONFLICTS} \subseteq I$. Hence, it is clear that none of the above conditions hold and the dealer is not discarded. We thus have the following two cases:

1. **There exists an honest party that outputs $(\text{detect}, \text{Bad})$:** In this case, it must hold that $|\text{Bad}| > t/2$. Since the corresponding message was broadcasted by the dealer, it must hold that all the honest parties output $(\text{detect}, \text{Bad})$. Since the simulator emulates the interaction of the honest parties with the dealer as in the real execution, all

the simulated honest parties hold the same set Bad . In that case, the simulator sends $(\text{detect}, \text{Bad})$ to the functionality \mathcal{F} , causing all the honest parties in the ideal world to output the same.

2. **There exists an honest party that outputs proceed :** This implies that $|\text{Bad}| \leq t/2$. Since this set was broadcasted by the dealer, all the honest parties hold the same set. Moreover, an honest party never belongs to CONFLICTS . Thus, we have that all the honest parties P_j output $(\text{proceed}, f_j(x), g_j(y))$, where $f_j(x)$ and $g_j(y)$ are consistent with a bivariate polynomial $S(x, y)$ by our input assumption. Since the simulator emulates the interaction of the honest parties with the dealer as in the real execution, all the simulated honest parties hold the same set Bad . In this case, the simulator sends proceed to the functionality \mathcal{F} , causing all the honest parties in the ideal world to output proceed . This is identical to the output of the honest parties in the real world.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z . Set $\text{HAVE-SHARES} = [n] \setminus (\text{ZEROS} \cup \text{CONFLICTS})$.
2. Receive from the functionality the sets ZEROS , CONFLICTS and the bivariate polynomial $S(x, y)$.
3. Simulate the protocol where each honest party P_j with $j \notin \text{CONFLICTS}$ starts with input $S(x, j), S(j, y)$, and all parties have the same sets CONFLICTS , ZEROS .
4. Send the message M to the functionality according to the following cases (the proof will show that the cases are mutually-exclusive):
 - (a) If the output of some simulated honest party is discard , then send discard to the functionality and halt.
 - (b) If the output of some simulated honest party is $(\text{detect}, \text{Bad})$, then send $(\text{detect}, \text{Bad})$ to the functionality and halt.
 - (c) If the output of some simulated honest party is proceed , then send proceed to the functionality and halt.

Since the simulator uses the exact same inputs of the simulated honest parties as the real honest parties in the real execution, and since the protocol is deterministic, we get that the view of the adversary is exactly the same in the real and in the ideal.

We now turn to show that the outputs of all honest parties is the same in the real and in the ideal, conditioned on the view of the adversary.

1. **There exists an honest party that outputs `discard` in the real world:** An honest party outputs `discard` in one of the following cases:
 - (a) The dealer broadcasted `Bad` such that (i) $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| > t$; or (ii) $\text{Bad} \not\subset \text{HAVE-SHARES}$. Since all honest parties hold the same sets `CONFLICTS` and `ZEROS`, and since `Bad` is broadcasted, we get that all honest parties would output `discard`.
 - (b) The dealer broadcasted `Bad` with $|\text{Bad}| \leq t/2$, and for which $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| \leq t$ and $\text{Bad} \subset \text{HAVE-SHARES}$. Moreover, some honest party $j \in \text{CONFLICTS}$ broadcasted `reveal(j)`, and when considering all points $R_j = \{(i, u_i)\}$ such that `reveal(i, j, u_i)` was broadcasted in Step 2b, and $i \in \text{HAVE-SHARES} \setminus \text{Bad}$, or $u_i = 0$ if $i \in \text{ZEROS}$, it holds that R_j does not define a unique polynomial of degree $t + d$. Since the set R_j is public, all honest parties will identify that there is no unique reconstruction, and all would output `discard`.

In both cases, in the ideal execution the simulator also sends `discard` to the functionality and all honest parties output `discard` in the ideal execution.

2. **There exists an honest party that outputs `(detect, Bad)`.** An honest party outputs `(detect, Bad)` if the dealer broadcast `Bad` such that $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| \leq t$, and $\text{Bad} \subset \text{HAVE-SHARES}$. Moreover, it holds that $|\text{Bad}| > t/2$. Since the message `Bad` is broadcast, then all honest parties would output `(detect, Bad)`. In the simulated execution we will have that the simulator sends to the functionality the message `(detect, Bad)`, and all honest parties output `(detect, Bad)`.
3. **There exists an honest party that outputs `proceed`.** First, if no honest party outputs `discard` and `(detect, Bad)`, then there must be an honest party in `HAVE-SHARES` that outputs `proceed`. We now claim that all honest parties not in `CONFLICTS` output `(proceed, $S(x, j), S(j, y)$)`, and all honest parties in `CONFLICTS` outputs `(proceed, $\perp, S(j, y)$)`, where $S(x, y)$ is the bivariate polynomial that is interpolated by the input shares of the honest parties and is guaranteed to exist under our input assumption.
 - (a) **All honest parties P_j with $j \notin \text{CONFLICTS}$ output `(proceed, $S(x, j), S(j, y)$)`.** Since an honest party did not output `discard` and `(detect, Bad)`, we have that $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| \leq t$, $\text{Bad} \subset \text{HAVE-SHARES}$, and $|\text{Bad}| \leq t/2$. Moreover, for every $j \in \text{CONFLICTS}$ that broadcast `complaint(j)`, the points R_j that were broadcasted (excluding the parties in `Bad`) define a unique polynomial. Since all the messages are broadcast, if one honest party P_k not in `CONFLICTS` out-

put $(\text{proceed}, S(x, k), S(k, y))$ then all honest parties P_j not in CONFLICTS have the exact same public view and also output $(\text{proceed}, S(x, j), S(j, y))$. Note that the honest parties P_j not in CONFLICTS hold their respective $S(x, j)$ and $S(j, y)$ polynomials consistent with a bivariate polynomial $S(x, y)$ according to our input assumptions.

- (b) **Each honest party P_j with $j \in \text{CONFLICTS}$ outputs $(\text{proceed}, \perp, S(j, y))$.** Here we have two sub-cases to consider. First, if in Step 2b the party P_j has a unique reconstruction, then the reconstruction must be $S(j, y)$. Specifically, let $g_j(y)$ be the unique reconstructed polynomial of P_j . It must hold that $S(j, k) = g_j(k)$ for at least $t + 1 + t/2 + d$ values of k , since each honest party not in CONFLICTS sent to P_j a point on $S(x, y)$. Since $g_j(y)$ is of degree $t + d$, and since $S(j, y)$ is also of degree $t + d$, we must have that $g_j(y) = S(j, y)$.
 Second, if in Step 2b the party P_j did not have a unique reconstruction, then it broadcast $\text{reveal}(j)$. It will receive its polynomial only in Step 5. There must be a unique reconstruction, as otherwise no honest party would have output proceed . We now claim that the unique reconstruction must be $S(j, y)$. As before, all honest parties broadcast values on the polynomial $S(j, y)$. The reconstruct polynomial therefore must agree with $S(j, y)$ on at least $t + 1 + t/2 + d$ points, and since this is a polynomial of degree $t + d$, the two polynomials must be identical. We conclude that P_j outputs $(\text{proceed}, \perp, S(j, y))$.

To conclude, in the simulated execution the simulator would submit to the functionality the message proceed . Each honest party $j \in \text{CONFLICTS}$ would output $(\text{proceed}, \perp, S(j, y))$, whereas each honest party $j \notin \text{CONFLICTS}$ would output $(\text{proceed}, S(x, j), S(j, y))$. This is exactly as in the real execution.

□

4.4.3 Reconstruction of f -Polynomials in CONFLICTS

The goal of this step is to make each party in CONFLICTS to receive its f -share. This is performed in a similar manner to that of reconstruction of g . This time, all honest parties hold shares of g , and thus each party in CONFLICTS receives at least $2t + 1$ correct values on each its f polynomial. The f -polynomial is of degree $3t/2$, and therefore we fail to reconstruct if the adversary introduces more than $t/2$ errors. In that case, we will have detection, in a similar manner to the reconstruction of g . The full details of the functionality (denoted by $\mathcal{F}_{\text{rec-}f}$), the protocol (denoted by $\Pi_{\text{rec-}f}$), and the proof are given below.

Functionality 4.4.7: Reconstruction of f -Polynomials – $\mathcal{F}_{\text{rec-f}}$

1. **Input:** All honest parties send to the functionality the sets $\text{ZEROS} \subset [n]$ and $\text{CONFLICTS} \subset [n]$, each honest party P_j for $j \notin (\text{CONFLICTS} \cup I)$ sends $(f_j(x), g_j(y))$. Each honest P_j for $j \in \text{CONFLICTS}$ sends $g_j(y)$. Let $S(x, y)$ be the unique bivariate polynomial of degree $3t/2$ in x and $t + d$ in y that satisfies $f_j(x) = S(x, j)$ and $g_j(y) = S(j, y)$ for every $j \notin \text{CONFLICTS}$ and $g_j(y) = S(j, y)$ for every $j \in \text{CONFLICTS}$. Moreover, it holds that $n - |\text{CONFLICTS}| \geq 2t + 1 + t/2 + d$.
 2. Send $(\text{ZEROS}, \text{CONFLICTS}, (S(x, i), S(i, y))_{i \in I})$ to the adversary. If the dealer is corrupted, then send also $S(x, y)$.
 3. Receive back from the adversary a message M .
 4. **Output:**
 - (a) If $M = \text{discard}$ and the dealer is corrupted, then send discard to all parties.
 - (b) If $M = (\text{detect}, \text{Bad})$ with $\text{Bad} \cap \text{ZEROS} = \emptyset$ and $|\text{Bad}| > t/2$, then send $(\text{detect}, \text{Bad})$ to all parties.
 - (c) If $M = \text{proceed}$ then send for each j the output $(\text{proceed}, S(x, j), S(j, y))$.
-

Protocol 4.4.8: Reconstruct f -Polynomials in CONFLICTS – $\Pi_{\text{rec-f}}$

- **Input:** All parties hold the same set CONFLICTS and ZEROS . Each honest party not in CONFLICTS holds a pair of polynomials $(f_i(x), g_i(y))$, and each honest party in CONFLICTS holds the polynomial $g_i(y)$. It is guaranteed that all the shares of honest parties lie on the same bivariate polynomial $S(x, y)$ with degree at most $3t/2$ in x and $t + d$ in y .
- **The protocol:**
 1. Set $\text{HAVE-SHARES} = [n] \setminus \text{ZEROS}$.
 2. For every $j \in \text{CONFLICTS}$:
 - (a) Each party P_i for $i \in \text{HAVE-SHARES}$ sends $(i, g_i(j))$ to P_j .
 - (b) Let (i, u_i) be the value P_j received from P_i . Moreover, for every $i \in \text{ZEROS}$, consider (i, u_i) with $u_i = 0$. Given all (i, u_i) , P_j looks for a codeword of distance at most $t/2$ from all the values it received. If there is such a codeword, set $f_j(x)$ to be the unique Reed-Solomon reconstruction (see Corollary 4.3.3, item 1). If there is no such a unique codeword, then P_j broadcasts $\text{complaint}(j)$ and every party P_i for $i \in \text{HAVE-SHARES}$ broadcasts $\text{reveal}(i, j, g_i(j))$.

3. The dealer sets $\text{Bad} = \emptyset$. For each $\text{reveal}(i, j, u)$ message broadcasted, the dealer verifies that $u = g_i(j)$. If not, then it adds i to Bad . The dealer broadcasts Bad .
 4. Verify that (i) $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| \leq t$; and (ii) $\text{Bad} \subset \text{HAVE-SHARES}$. Otherwise, discard – go to Step 7a.
 5. If $|\text{Bad}| > t/2$ then there is a large detection – go to Step 7b.
 6. Otherwise, for every $j \in \text{CONFLICTS}$, if $\text{complaint}(j)$ was broadcasted, then consider all the points $R_j = \{(i, u_i)\}$ such that $\text{reveal}(i, j, u_i)$ was broadcasted in Step 2b, and $i \in \text{HAVE-SHARES} \setminus \text{Bad}$, or $u_i = 0$ if $i \in \text{ZEROS}$. Verify that R_j defines a unique polynomial of degree $3t/2$. If not, go to Step 7a. Otherwise, P_j sets $f_j(x)$ to be that unique polynomial.
 7. **Output:**
 - (a) Discard the dealer: Output discard.
 - (b) Detect: Output $(\text{detect}, \text{Bad})$.
 - (c) Proceed: Each party j outputs $(\text{proceed}, f_j(x), g_j(y))$.
-

Lemma 4.4.9. *The Protocol $\Pi_{\text{rec-f}}$ (Protocol 4.4.8), perfectly securely computes the $\mathcal{F}_{\text{rec-f}}$ functionality (Functionality 4.4.7), in the presence of a malicious adversary, controlling at most $t < n/3$.*

Proof. We show the case of an honest dealer and a corrupted dealer separately.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z . Set $\text{HAVE-SHARES} = [n] \setminus \text{ZEROS}$.
2. Receive from the functionality the sets $\text{ZEROS}, \text{CONFLICTS}$ and the polynomials $S(x, i), S(i, y)$ for every $i \in I$.
3. Set $\text{Bad} = \emptyset$. For every $i \in \text{CONFLICTS}$: (note that since the dealer is honest, then $\text{CONFLICTS} \subseteq I$. Thus, any such element is also in I)
 - (a) Simulate party P_j , $j \in \text{HAVE-SHARES}$ sending $(j, i, g_j(i)) = (j, i, f_i(j))$ to the adversary.
 - (b) If P_i broadcasts $\text{complaint}(i)$ then simulate party P_j , for $j \in \text{HAVE-SHARES}$ broadcasting $\text{reveal}(j, i, f_i(j))$.
 - (c) Listen to all broadcasts $\text{reveal}(i', i, u_i)$ that the adversary sends for some $i' \in \text{HAVE-SHARES} \cap I$. If $u_i \neq f_i(i')$, then add i' to Bad .

4. Simulate the dealer broadcasting Bad . If $|\text{Bad}| > t/2$, then send $(\text{detect}, \text{Bad})$ to the functionality and halt.
5. Otherwise, send proceed to the functionality, and halt.

The protocol as well as the simulator is deterministic. Hence, it can be easily observed that the view of the adversary in the real and ideal executions is identical. It remains to show that the output of honest parties is identical in the real and ideal world.

Towards that end, note that in the real world, an honest dealer is discarded if and only if one of the following conditions holds:

1. $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| > t$.
2. $\text{Bad} \not\subseteq \text{HAVE-SHARES}$.
3. R_j does not define a unique polynomial of degree- $3t/2$.

Note that for an honest dealer, an honest party never belongs to Bad and we have that $\text{ZEROS} \subseteq I$ and $\text{CONFLICTS} \subseteq I$. It is clear that none of the above conditions hold and the dealer is not discarded. We thus have the following two cases:

1. **There exists an honest party that outputs $(\text{detect}, \text{Bad})$:** In this case, it must hold that $|\text{Bad}| > t/2$. Since the corresponding message was broadcasted by the dealer, it must hold that all the honest parties output $(\text{detect}, \text{Bad})$. The simulator emulates the interaction of the honest parties with the dealer as in the real execution, hence all the simulated honest parties hold the same set Bad . In that case, the simulator sends $(\text{detect}, \text{Bad})$ to the functionality, causing all the honest parties in the ideal world to output the same.
2. **There exists an honest party proceed :** This implies that $|\text{Bad}| \leq t/2$. Since this set was broadcasted by the dealer, all the honest parties hold the same set. Moreover, an honest party never belongs to CONFLICTS . Thus, we have that all the honest parties P_j output $(\text{proceed}, f_j(x), g_j(y))$ where $f_j(x)$ and $g_j(y)$ are consistent with the bivariate polynomial $S(x, y)$ as guaranteed by our input assumption. As mentioned, the simulator emulates the interaction of the honest parties with the dealer as in the real execution, hence all the simulated honest parties hold the same set Bad . In this case, the simulator sends proceed to the functionality, causing all the honest parties P_j in the ideal world to output $(\text{proceed}, S(x, j), S(j, y))$. This is identical to the output of the honest parties in the real world.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z . Set $\text{HAVE-SHARES} = [n] \setminus \text{ZEROS}$.
2. Receive from the functionality the sets ZEROS , CONFLICTS and the bivariate polynomial $S(x, y)$.
3. Simulate the protocol where each honest party P_j with $j \notin \text{CONFLICTS}$ starts with input $S(x, j), S(j, y)$, each honest P_j with $j \in \text{CONFLICTS}$ starts with input $S(j, y)$ and all parties have the same sets CONFLICTS , ZEROS .
4. Send the message M to the functionality according to the following cases (the proof will show that the cases are mutually-exclusive):
 - (a) If the output of some simulated honest party is `discard`, then send `discard` to the functionality and halt.
 - (b) If the output of some simulated honest party is `(detect, Bad)`, then send `(detect, Bad)` to the functionality and halt.
 - (c) If the output of some simulated honest party is `proceed`, then send `proceed` to the functionality and halt.

Since the simulator uses the exact same inputs of the simulated honest parties as the real honest parties in the real execution, and since the protocol is deterministic, we get that the view of the adversary is exactly the same in the real and in the ideal. Thus it remains to be shown that the output of the honest parties is identical in the real and ideal executions. We have the following cases to consider:

1. **There exists an honest party that outputs `discard` in the real world:** Observe that an honest party outputs `discard` if and only if one of the following conditions holds.
 - (a) The dealer broadcasted `Bad` such that either (i) $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| > t$; or (ii) $\text{Bad} \not\subset \text{HAVE-SHARES}$. Since all honest parties hold the same sets CONFLICTS and ZEROS , and the set `Bad` is broadcasted, we get that all honest parties output `discard`.
 - (b) The dealer broadcasted `Bad` with $|\text{Bad}| \leq t/2$, and for which $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| \leq t$ and $\text{Bad} \subset \text{HAVE-SHARES}$. Moreover, some honest party $j \in \text{CONFLICTS}$ broadcasted `reveal(j)`, and when considering all points $R_j = \{(i, u_i)\}$ such that `reveal(i, j, u_i)` was broadcasted in Step 2b, and $i \in \text{HAVE-SHARES} \setminus \text{Bad}$, or $u_i = 0$ if $i \in \text{ZEROS}$, it holds that R_j does not define a unique polynomial of degree $3t/2$. Since the set R_j is public, all honest parties will identify that there is no unique reconstruction, and all would output `discard`.

Since the simulated honest parties have the same view as the honest parties, the simulated honest parties also output `discard`. In this case, the simulator sends `discard` to

- the functionality causing all the honest parties to output `discard` in the ideal execution.
2. **There exists an honest party that outputs `(detect, Bad)` in the real world:** In this case, it must hold that $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| \leq t$ and $\text{Bad} \subset \text{HAVE-SHARES}$. Moreover, it must hold that $|\text{Bad}| \geq t/2$. Since the corresponding set `Bad` is broadcast, all the honest parties hold the same set and hence output `(detect, Bad)`. The simulated honest parties hold an identical output, and thus the simulator sends `(detect, Bad)` to the functionality, which in turn sends the same to all the honest parties in the ideal execution.
 3. **There exists an honest party that outputs `proceed` in the real world:** In this case, we show that each honest P_j outputs `(proceed, $S(x, j)$, $S(j, y)$)` where $S(x, y)$ is the bivariate polynomial that is interpolated from the input shares of the honest parties and is guaranteed to exist under our input assumption. We consider the following two cases.
 - (a) **Each honest P_j with $j \notin \text{CONFLICTS}$ outputs `(proceed, $S(x, j)$, $S(j, y)$)`:** Since there exists an honest party that does not output `discard` or `(detect, Bad)`, it must hold that $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| \leq t$, $\text{Bad} \subset \text{HAVE-SHARES}$ and $|\text{Bad}| \leq t/2$. Moreover, for every $j \in \text{CONFLICTS}$ which broadcast `complaint(j)`, the points R_j which were broadcast (excluding the points of parties in `Bad`) define a unique degree- $3t/2$ polynomial. Since all the corresponding messages were broadcast, all the honest parties not in `CONFLICTS` have the same view and hence output `(proceed, $S(x, j)$, $S(j, y)$)`, where the polynomials $S(x, j)$ and $S(j, y)$ are consistent with $S(x, y)$ and held by the parties not in `CONFLICTS` according to our input assumption.
 - (b) **Each honest P_j with $j \in \text{CONFLICTS}$ outputs `(proceed, $S(x, j)$, $S(j, y)$)`:** Here we have two sub-cases to consider. First, note that if P_j has a unique reconstruction in Step 2b, then the reconstruction must be $S(x, j)$. Specifically, let $f_j(x)$ be the unique degree- $3t/2$ reconstructed polynomial of P_j . It must hold that $S(k, j) = f_j(k)$ for at least $2t + 1$ values of k , since each honest party P_k sent to P_j a point on $S(x, j)$. Since both $f_j(x)$ and $S(x, j)$ are of degree $3t/2$, we have that $f_j(x) = S(x, j)$ holds. Second, if in Step 2b the party P_j did not have a unique reconstruction, then it must be that P_j broadcast `reveal(j)`. It will thus receive its polynomial only in Step 6. In this case, it must hold that there is a unique reconstruction of degree- $3t/2$ polynomial from the publicly revealed points in R_j which were broadcast (excluding the points of parties in `Bad`), as otherwise no honest party would have output `proceed`. We now claim that the unique reconstruction must be $S(x, j)$. As

before, all honest parties broadcast values on the polynomial $S(x, j)$. Moreover, since $|\text{Bad}| \leq t/2$ (otherwise parties would output $(\text{detect}, \text{Bad})$), it must hold that the reconstructed polynomial agrees with $S(x, j)$ on at least $3t/2 + 1$ points of the honest parties. Since both the reconstructed polynomial and $S(x, j)$ are of degree $3t/2$, the two polynomials must be identical. We conclude that P_j outputs $(\text{proceed}, S(x, j), S(j, y))$, where it is guaranteed to hold $S(j, y)$ due to our input assumption.

Consequently, in the simulated execution the simulator sends proceed to the functionality. Each honest party j thus outputs $(\text{proceed}, S(x, j), S(j, y))$ in the ideal execution. This is exactly as in the real execution. □

4.4.4 Putting Everything Together: Packed Secret Sharing

We view a list of $(t/2 + 1)(d + 1)$ secrets SECRETS as a $(t/2 + 1) \times (d + 1)$ matrix.

Functionality 4.4.10: Packed Secret Sharing – \mathcal{F}_{PSS}

The functionality is parameterized by the set of corrupted parties $I \subseteq [n]$.

- **Input:** All parties input a set $\text{ZEROS} \subset [n]$ such that $|\text{ZEROS}| \leq t$. If the dealer is honest then it is guaranteed that $\text{ZEROS} \subseteq I$.
 - **Honest dealer:** The dealer sends SECRETS to \mathcal{F}_{PSS} . The functionality sends ZEROS to the adversary, which replies with $(f_i(x), g_i(y))_{i \in I}$ under the constraint that $f_i(x) = g_i(y) = 0$ for every $i \in \text{ZEROS}$. The functionality chooses a random bivariate polynomial $S(x, y)$ of degree $3t/2$ in x and $t + d$ in y under the constraints that (i) SECRETS is embedded in S (see Section 4.3 for the meaning of embedding); (ii) $S(x, i) = f_i(x)$ for every $i \in I$; (iii) $S(i, y) = g_i(y)$.
 - **Corrupted dealer:** The functionality sends ZEROS to the adversary, which replies with $S(x, y)$. \mathcal{F}_{PSS} verifies that $S(x, y)$ is of degree $3t/2$ in x and degree $t + d$ in y , and that for every $i \in \text{ZEROS}$ it holds that $f_i(x) = g_i(y) = 0$. If not, \mathcal{F}_{PSS} replaces $S(x, y) = \perp$.
 - **Output:** \mathcal{F}_{PSS} sends to each party P_j the pair of polynomials $S(x, j), S(j, y)$.
-

We claim that there is always a bivariate polynomial that can be reconstructed. Specifically, consider for simplicity the case where $|I| = t$:

1. A bivariate polynomial of degree $3t/2$ in x and degree $t + d$ in y is determined by $(3t/2 + 1)(t + d + 1)$ values.

2. The adversary sends t pairs of polynomials of degree $3t/2$ and $t + d$. The f polynomials define $t(3t/2 + 1)$ values. Each g polynomial is already determined in t coordinates, and therefore we have a total of $t(t + d + 1 - t) = t(d + 1)$.
3. SECRETS determines $(t/2 + 1) \cdot (d + 1)$ values.

Therefore, the number of constraints that we have is $(t/2 + 1)(d + 1) + t(3t/2 + 1) + t(d + 1)$, which is exactly $(3t/2 + 1)(t + d + 1)$, the total number of variables in the bivariate polynomial.

Protocol 4.4.11: Packed Secret Sharing in the $(\mathcal{F}_{\text{ShareAttempt}}, \mathcal{F}_{\text{rec-g}}, \mathcal{F}_{\text{rec-f}})$ -hybrid model – Π_{PSS}

Input: The dealer holds SECRETS, and all honest parties hold the same set ZEROS.

The protocol:

1. Dealing the shares:

- (a) The dealer chooses a random bivariate polynomial $S(x, y)$ of degree at most $3t/2$ in x and degree $t + d$ in y that embeds SECRETS, under the constraint that for every $i \in \text{ZEROS}$ it holds that $S(x, i) = 0$ and $S(i, y) = 0$.
- (b) All parties invoke Functionality 4.4.1, $\mathcal{F}_{\text{ShareAttempt}}$, where the dealer inputs $S(x, y)$ and all parties input ZEROS:
 - i. If the output is `discard`, then proceed to Step 4a.
 - ii. If the output is `(detect, CONFLICTS)` then set $\text{ZEROS} = \text{ZEROS} \cup \text{CONFLICTS}$. If $|\text{ZEROS}| > t$ then proceed to Step 4a. Otherwise, go back to Step 1a.
 - iii. If the output is `(proceed, $f_i(x), g_i(y)$, CONFLICTS)`, then proceed to the next step. Note that it must hold that (a) for parties $i \in \text{CONFLICTS}$, $f_i(x) = g_i(y) = \perp$ and (b) $n - |\text{CONFLICTS}| \geq n - (t/2 - d)$.

2. Reconstruct the g -polynomials: The parties invoke Functionality 4.4.4, $\mathcal{F}_{\text{rec-g}}$, where each party P_i inputs $(\text{ZEROS}, \text{CONFLICTS}, f_i(x), g_i(y))$.

- (a) If the output is `discard`, then proceed to Step 4a.
- (b) If the output is `(detect, Bad)` then set $\text{ZEROS} = \text{ZEROS} \cup \text{Bad}$. If $|\text{ZEROS}| > t$ then discard and proceed to Step 4a. Otherwise, go back to Step 1a.
- (c) Otherwise, the output is `(proceed, $f_i(x), g_i(y)$)` where every party P_i with $i \in \text{CONFLICTS}$ has $g_i(y) \neq \perp$, then proceed to the next step.

3. Reconstruct the f -polynomials: The parties invoke Functionality 4.4.7, $\mathcal{F}_{\text{rec-f}}$, where each party P_i inputs $(\text{ZEROS}, \text{CONFLICTS}, f_i(x), g_i(y))$. Note that for parties in CONFLICTS it holds that $f_i(x) = \perp$.

- (a) If the output of the functionality is `discard`, then proceed to Step 4a.

- (b) If the output is (detect, Bad) then set $\text{ZEROS} = \text{ZEROS} \cup \text{Bad}$. If $|\text{ZEROS}| > t$ then discard and go to Step 4a. Otherwise, go back to Step 1a.
- (c) Otherwise, let (proceed, $f_i(x), g_i(y)$) be the output, where now all parties have $f_i(x) \neq \perp$ and $g_i(y) \neq \perp$. Go to Step 4b.

4. Output:

- (a) **Discard:** All parties output \perp .
 - (b) **Successful:** Output $f_i(x), g_i(y)$.
-

Lemma 4.4.12. *Let $t < n/3$ and $d \leq t/4$. Protocol 4.4.11, Π_{PSS} , perfectly securely computes Functionality 4.4.10, \mathcal{F}_{PSS} , in the $(\mathcal{F}_{\text{ShareAttempt}}, \mathcal{F}_{\text{rec-g}}, \mathcal{F}_{\text{rec-f}})$ -hybrid model (Functionality 4.4.1, 4.4.4, 4.4.7)), in the presence of a malicious adversary, controlling at most $t < n/3$.*

Proof. We separate between the case of an honest dealer and a corrupted dealer.

The case of an honest dealer. The simulator is as follows:

1. Invoke \mathcal{A} on an auxiliary input z .
2. Receive from the functionality the set ZEROS.
3. Set SECRETS arbitrarily as input (say, all zeros) and run the protocol where the dealer holds SECRETS and all other parties have ZEROS as input. In particular, simulate all inner functionalities as a functionality would run them.
4. Let $S(x, y)$ be the input of the simulated honest dealer used and sent to the simulated Functionality 4.4.1 in the last iteration (by iteration, we mean running the protocol from Step 1a until restarting or concluding the protocol). Send $S(x, i), S(i, y)$ to the functionality for every $i \in I$.

We now show that the output of the real and ideal executions are the same. Towards that end, consider the following games:

- **Game₁:** This is the real execution. We run the protocol where the honest dealer uses SECRETS as its input. The output of this experiment is the view of the adversary and the output of all honest parties in the protocol.
- **Game₂:** We run a modified ideal model, in which the simulator receives the same SECRETS as in Game₁ as an advice, and the dealer uses SECRETS as its input to the functionality. The simulator uses SECRETS as its input instead of all zeros as the description of \mathcal{S} . The simulator runs the protocol where the input of the honest dealer is

SECRETS, exactly as the real execution in Game_1 . We claim that the dealer is never discarded. Then, the simulator sends to the functionality the output shares of the corrupted parties in the simulated execution. The functionality chooses some random polynomial $S(x, y)$ that agrees with the output shares of the adversary and SECRETS, and gives the honest parties their shares on that polynomial. The output of this experiment is the view of the adversary as determined by the simulator, and the output of all honest parties (equivalent to $S(x, y)$).

- Game_3 : This is the ideal model. In particular, the simulator receives no advice, and runs as in Game_2 , but with input $\text{SECRETS} = 0$.

We show that all the outputs of all games are identically distributed.

The outputs of Game_1 and Game_2 are identically distributed. The simulator in Game_2 runs the exact same protocol as the real execution in Game_1 , and therefore the view of the adversary is identical in both executions. We now turn to the output of the honest parties. We claim that in that execution, the honest dealer is never discarded. Specifically:

1. The honest dealer chooses a bivariate polynomial that always satisfies the conditions of Functionality 4.4.1 and therefore that functionality never returns `discard`. Moreover, $\text{CONFLICTS} \subseteq I$, and therefore we never reach $|\text{ZEROS}| > t$ and the dealer is never discarded. Furthermore, we can reboot the protocol only a constant number of times (see Corollary 4.4.15).
2. When invoking Functionality 4.4.4 we have that the shares of the honest parties satisfy the input assumption of the functionality. Moreover, $\text{Bad} \subset I$, and so again $\text{Bad} \cup \text{CONFLICTS} \subseteq I$, and so the dealer is not discarded. Again, we can reboot the protocol only a constant number of times (see Corollary 4.4.15).
3. Finally, when invoking Functionality 4.4.7 we have that the shares of the honest parties satisfy the input assumption of the functionality. From a similar reasons as above, the output would be shares of the reconstructed polynomial, which is the same polynomial as the dealer used in the beginning of the iteration as its input to $\mathcal{F}_{\text{ShareAttempt}}$.

We conclude that when the dealer is honest, all parties output shares on the same bivariate polynomial, which is a polynomial $S(x, y)$ that the dealer used in that iteration. In Game_1 , the output of all honest parties is shares on that polynomial. In Game_2 , the simulator sends the shares $(S(x, i), S(i, y))_{i \in I}$ to the functionality, the functionality samples a new polynomial $S'(x, y)$ under the constraints that $S'(x, i) = S(x, i)$ and $S'(i, y) = S(i, y)$ for every $i \in I$, and SECRETS is embedded in $S'(x, y)$. The output of all honest parties is then equivalent to just outputting $S'(x, y)$. We claim that the output is identical via the following claim:

Claim 4.4.13. Let SECRETS be any arbitrary sets of $(t/2 + 1)(d + 1)$ field elements, and let $I \subset [n]$ be a set of cardinality at most t . Then, for every ZEROS $\subseteq I$ the output of the following two distributions is identical:

Process I

- Choose a random $(3t/2, t + d)$ -bivariate polynomial $S(x, y)$ that embeds SECRETS, such that for every $i \in \text{ZEROS}$ it holds that $S(x, i) = S(i, y) = 0$.
- Output $S(x, y)$.

Process II

- Choose a random $(3t/2, t + d)$ -bivariate polynomial $S(x, y)$ that embeds SECRETS, such that for every $i \in \text{ZEROS}$ it holds that $S(x, i) = S(i, y) = 0$.
- Choose a random $(3t/2, t + d)$ -bivariate polynomial $S'(x, y)$ that embeds SECRETS such that for every $i \in I$ it holds that $S'(x, i) = S(x, i)$ and $S'(i, y) = S(i, y)$.
- Output $S'(x, y)$.

Proof. For the case of $|I| = t$, we show that $S'(x, y) = S(x, y)$. Let $S(x, i) = f_i(x)$ and $S(i, y) = g_i(y)$.

We claim that there is a unique polynomial $S'(x, y)$ that can embed SECRETS and satisfies for every $i \in I$ the conditions $S'(x, i) = f_i(x)$ and $S'(i, y) = g_i(y)$. Specifically, reconstruct the polynomials $g_0(y), \dots, g_{-t/2}(y)$ of degree $t + d$ each as follows: $g_{-a}(-b) = \text{SECRETS}(a, b)$ for $a \in \{0, \dots, t/2\}$ and $b \in \{0, \dots, d\}$. Moreover, for every $a \in \{0, \dots, t/2\}$ and $i \in I$ we set $g_{-a}(i) = f_i(-a)$. This defines $t + d + 1$ points on each one of the polynomials $g_0(y), \dots, g_{-t/2}(y)$, and uniquely define polynomials of degree $t + d$. Now, from Lagrange interpolation there exists a unique bivariate polynomial $S'(x, y)$ of degree $3t/2$ in x and $t + d$ in y that satisfies $S'(a, y) = g_{-a}(y)$ for every $a \in \{0, \dots, t/2\}$ and $S'(i, y) = g_i(y)$ for every $i \in I$. Note that those are $t/2 + t + 1$ polynomials of degree $t + d$ each, and therefore uniquely define $S'(x, y)$. This polynomial embeds SECRETS, agrees with $g_i(y)$ for every $i \in I$, and also satisfies $S'(x, i) = f_i(x)$ for every $i \in I$, since $f_i(j) = g_j(i) = S(j, i)$ for every $i, j \in I$, and $f_i(-a) = g_{-a}(i) = S(-a, i)$ for every $i \in \{0, \dots, t/2\}$. Thus the two univariate polynomials, $S(i, y)$ and $f_i(y)$ of degree $3t/2$ must agree.

For the case of $|I| < t$, we can just view process I as first choosing polynomials $f_i(x), g_i(y)$ for every $i \in \text{ZEROS}$ such that $f_i(x), g_i(y) = 0$, and then choosing $f_i(x), g_i(y)$ for every $i \in I \setminus \text{ZEROS}$ uniformly at random under the constraint that $f_i(j) = g_j(i)$ for every $i, j \in I$. Finally, choose $S(x, y)$ uniformly at random under the constraint that $S(x, i) = f_i(x)$ and

$S_i(y) = g_i(y)$ for every $i \in I$. The two process are therefore equivalent. \square

Notice that Process I is equivalent to the choice of the polynomial $S(x, y)$ in Game₁. Process II is equivalent to Game₂.

The outputs of Game₂ and Game₃ are identically distributed. The only difference between the two games is that in Game₂ the simulator uses the same secret SECRETS as the honest dealer uses in the ideal execution, whereas in Game₃ the the simulator uses SECRETS = 0. The following claim shows that the shares that the corrupted parties receive in the simulated execution is identically distributed. In both execution, given the shares that the simulator sends to the functionality, the outputs of the honest parties are defined in exactly the same process (the functionality uses SECRETS and the shares sent by the adversary). Therefore it is enough to show that the view is identically distributed.

Claim 4.4.14. *Let SECRETS₁, SECRETS₂ be two arbitrary sets of $(t/2 + 1)(d + 1)$ field elements, and let $I \subset [n]$ be a set of cardinality at most t . Then, for every ZEROS $\subseteq I$ the output of the following two distributions is identical:*

Process I

- Choose a random $(3t/2, t + d)$ -bivariate polynomial $S_1(x, y)$ that embeds SECRETS₁, such that for every $i \in \text{ZEROS}$ it holds that $S_1(x, i) = S_1(i, y) = 0$.
- Output $(i, S_1(x, i), S_1(i, y))$ for every $i \in I$.

Process II

- Choose a random $(3t/2, t + d)$ -bivariate polynomial $S_2(x, y)$ that embeds SECRETS₂, such that for every $i \in \text{ZEROS}$ it holds that $S_2(x, i) = S_2(i, y) = 0$.
- Output $(i, S_2(x, i), S_2(i, y))$ for every $i \in I$.

Proof. We show that the probability distributions $\{\{(i, S_1(x, i), S_1(i, y))\}_{i \in I}\}$ corresponding to **Process I** and $\{\{(i, S_2(x, i), S_2(i, y))\}_{i \in I}\}$ corresponding to **Process II** are identical. Towards that end, we begin by defining the probability ensembles \mathbb{S}_2 and \mathbb{S}_1 as follows:

$$\mathbb{S}_1 = \{\{(i, S_1(x, i), S_1(i, y))\}_{i \in I} \mid S_1 \text{ embeds SECRETS}_1 \text{ and } S_1(x, i) = S_1(i, y) = 0 \forall i \in \text{ZEROS}\}$$

$$\mathbb{S}_2 = \{\{(i, S_2(x, i), S_2(i, y))\}_{i \in I} \mid S_2 \text{ embeds SECRETS}_2 \text{ and } S_2(x, i) = S_2(i, y) = 0 \forall i \in \text{ZEROS}\}$$

Given this, we show that $\mathbb{S}_1 \equiv \mathbb{S}_2$. For this, we show that given any set of pairs of degree- $3t/2$ and degree- t polynomials $Z = \{f_i(x), g_i(y)\}_{i \in I}$ that satisfy $f_i(j) = g_j(i)$ for every $i, j \in I$, the number of bivariate polynomials in support of \mathbb{S}_1 that are consistent with Z are the same as the number of polynomials in support of \mathbb{S}_2 .

First, observe that if there exist $i, j \in I$ such that $f_i(j) \neq g_j(i)$, or if there exists some $i \in I \cap \text{ZEROS}$ such that $f_i(x) \neq 0$ or $g_i(y) \neq 0$, then there does not exist any bivariate polynomial in support of \mathbb{S}_1 or \mathbb{S}_2 that is consistent with Z .

Now consider $Z = \{f_i(x), g_i(y)\}_{i \in I}$ such that for every $i, j \in I$, it holds that $f_i(j) = g_j(i)$. Moreover, for each $i \in I \cap \text{ZEROS}$, it holds that $f_i(x) = g_i(y) = 0$. We begin by counting the number of polynomials in support of \mathbb{S}_1 that are consistent with Z . For the case when $|I| = t$, note that Z together with SECRETS_1 defines exactly one polynomial $S(x, y)$, as we saw in the proof of Claim 4.4.13.

When $|I| < t$, we can first define $g_{-a}(y)$ for each $a \in \{0, \dots, t/2\}$ by choosing $t - |I|$ points on each of these polynomials uniformly at random (since $g_{-a}(y)$ is degree- $(t + d)$ polynomial, of which $|I|$ points are already defined by $\{f_i(-a)\}_{i \in I}$ and additionally $d + 1$ points are defined by $\text{SECRETS}_1(a, b)$ for each $b \in \{0, \dots, d\}$). Following this, the number of polynomials in the support is $|\mathbb{F}|^{(t/2+1)(t-|I|)}$.

A similar argument shows the same calculation for choosing S_2 according to \mathbb{S}_2 . Finally, since S_1 and S_2 are chosen randomly from those consistent with Z and SECRETS_1 or SECRETS_2 respectively, the probability that Z is obtained is same in both the cases. \square

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} on the auxiliary input z .
2. Receive from the functionality the set ZEROS .
3. Simulate running the protocol with the adversary when all honest parties hold ZEROS as input, while also simulating Functionalities 4.4.1, 4.4.4, 4.4.7 to the adversary.
4. If the output of some simulated honest party is \perp , then send $S(x, y) = x^{3t/2+1}$ to the functionality, in which case it sends \perp to all parties.
5. Otherwise, let J be a set of $t + d + 1$ honest parties. Reconstruct the unique bivariate polynomial $S(x, y)$ that satisfies $S(x, j) = f_j(x)$ for every $j \in J$, where $f_j(x), g_j(y)$ is the output of the simulated honest party in the simulated execution. Send $S(x, y)$ to the functionality and halt.

Since the code of each party in the protocol which is not the dealer is deterministic, and the functionalities 4.4.1, 4.4.4, 4.4.7 are also deterministic, the view of the adversary is *identical* in the real and ideal. Moreover, since the functionality is deterministic, we can separately consider the view and the outputs of the honest parties. All is left is to show that the output of the honest parties is the same in the real and in the ideal executions. We have two cases to consider:

1. **There exists an honest party that outputs \perp .** That is, the shares are rejected. This happens if one of the functionalities returns `discard`, in which case all honest parties receive the same output and all honest parties output \perp . Or, it might be the case that the output of one of the invocations of Functionality 4.4.1 is `(detect, CONFLICTS)` and it holds that $\text{ZEROS} = \text{ZEROS} \cup \text{CONFLICTS}$ satisfies $|\text{ZEROS}| > t$. Again, since the output of all parties is the same, and their view of the sets `ZEROS` and `CONFLICTS` is the same, all honest parties output \perp . In the ideal execution, the simulated honest parties would also have the exact same output. In that case, the simulator sends $x^{3t/2+1}$, and the functionality delivers \perp to all honest parties. We conclude that in that case the output of the honest parties is \perp both in the real and in the ideal.
2. **One honest party did not output \perp .** In that case, we claim that no honest party outputs \perp . This is similar to the previous case. Moreover, by the guarantee of functionalities 4.4.1, 4.4.4, 4.4.7 we have that all honest parties have shares of the same bivariate polynomial $S(x, y)$. In the ideal execution, the simulated honest parties would also have shares of that polynomial $S(x, y)$. The simulator chooses an arbitrary set J of $t + d + 1$ honest parties, and reconstructs the unique bivariate polynomial that agree with their outputs. It must hold that this polynomial is $S(x, y)$. It sends this polynomial to the functionality, and each honest party receives as output the shares $S(x, i), S(i, y)$, exactly as in the real.

□

Communication and Efficiency Analysis. We conclude the following lemma, proven subsequently:

Lemma 4.4.15. *Let $t < n/3$ and $d \leq t/4$. There exists a protocol that implements Functionality 4.4.10, has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast for sharing $\mathcal{O}((d + 1)n)$ values (i.e., $\mathcal{O}(n(d + 1) \log n)$ bits) simultaneously in $\mathcal{O}(1)$ rounds. Every party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

Proof. By combining Theorems 4.4.3, 4.4.6, 4.4.9, and 4.4.12, we conclude the existence of a protocol that securely computes Functionality 4.4.10 in the plain model.

Further, Protocol 4.4.11 requires a constant number of restarts before it terminates successfully. To see this, we analyze the case of an honest dealer and corrupt dealer separately.

In the case of an honest dealer, note that a restart may occur at Steps 1(b)ii, 2b or 3b in Protocol 4.4.11. In each of the above cases, due to guarantees of functionality 4.4.1, 4.4.4

and 4.4.7, it is ensured that a new CONFLICTS set of cardinality more than $t/2 - d$, $t/2$ and $t/2$ respectively is identified. Since $d \leq t/4$, we have that more than $t/4$ conflicts are identified each time that the protocol requires a restart. Recall that in the case of the honest dealer, it is guaranteed that no honest party belongs to CONFLICTS. Thus, each time a CONFLICTS set is identified, a new set of corrupt parties is identified, which are publicly emulated (by setting their shares to 0) in the subsequent run. Consequently, after (at most) 3 restarts, it is guaranteed that more than $3t/4 \geq t/2 + d$ corrupt parties are emulated publicly and hence honestly. Since the number of parties in conflict with the honest dealer can be at most t , we have that at most $t - 3t/4$, that is $< t/4$ parties can misbehave in the subsequent execution of the protocol. Given that each functionality 4.4.1, 4.4.4 and 4.4.7 succeeds in this case, we have that Protocol 4.4.11 successfully terminates.

For a corrupt dealer, if the protocol terminates after (at most) 3 restarts, then by the guarantees of the protocol, we have that it computes Functionality 4.4.10. Otherwise, we are guaranteed that the dealer is corrupt, and hence can be discarded. The protocol requires the transmission of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels, and each party broadcasts at most $\mathcal{O}(n \log n)$ bits. \square

4.5 Batched and Packed Secret Sharing

In this section, we suggest how to keep the broadcast unchanged when running m instances of the packed secret sharing with the same dealer. That is, if one instance requires $\mathcal{O}(n^2 \log n)$ bits communicated over point-to-point channels and each party (including the dealer) broadcasts $\mathcal{O}(n \log n)$ bits, we have a protocol that requires $\mathcal{O}(mn^2 \log n)$ bits communicated over point-to-point channels and each party still has to broadcast at most $\mathcal{O}(n \log n)$ bits (and a total of $\mathcal{O}(n^2 \log n)$). We review the changes necessary for each one of the sub-protocols of packed secret sharing.

Sharing attempt and Batched Complaints. Here the dealer inputs m bivariate polynomials, but there is *one* set $\text{ZEROS} \subset [n]$. It is assumed that all bivariate polynomials have 0 shares for the parties in ZEROS.

At Step 2b in Protocol 4.4.2, every P_i checks consistency in all instances but raises a complaint for only one of them, say, the minimum index of the instance. A complaint now looks like $\text{complaint}(i, j, f_i(j), g_i(j), \alpha)$ where $\alpha \in \{1, \dots, m\}$. Moreover, if a party broadcasts $\text{complaint}(i, j, u_i, v_i)$ for $j \in \text{ZEROS}$, then the dealer must add P_i to CONFLICTS. Thus, there is no need for P_i to broadcast such a complaint in each instance that it sees inconsistency with P_j for $j \in \text{ZEROS}$, but it is enough to do it in only one of the instances.

This keeps the broadcast cost $\mathcal{O}(n^2 \log n)$ bits among all m instances combined (as opposed $\mathcal{O}(mn^2 \log n)$ when running them simultaneously in a black-box manner).

Note that when the dealer is honest, honest parties never complain on one another, and this holds in all m invocations. Moreover, if the dealer is corrupted and two honest parties have to file a joint complaint, then both will have the exact same minimal index, and the dealer must have to add one of them into CONFLICTS, exactly as we have in single instance.

Batched reconstruction of g polynomials in CONFLICTS. Here the change in the protocol is more delicate than the previous case, and we provide a full modeling and proof. Specifically, In Step 2b of Protocol 4.4.5, a party P_j may fail to reconstruct g_j in multiple instances. However, it is enough to pick one instance β (say, the one with minimum index) and complains publicly with β . Now, rest of the public verification happens with respect to β th invocation. If parties publicly reveal values that are different than what they revealed privately, then the party knows that those parties are corrupted and can try to reconstruct the polynomials without those shares. In particular, the only case when a party cannot uniquely reconstruct is when the the adversary introduces more than $t/2$ errors. However, if the public reconstruction of g in the β th execution is successful, it can recognize $t/2$ misbehaving parties by comparing the polynomial that was publicly reconstruct to the shares sent to it privately. Note that it is possible that a corrupted party sends some share to P_j privately but makes some other value public. P_j knows for sure that such party is corrupt, even though Bad that the dealer broadcasts can even be empty. Once P_j recognizes more than $t/2$ errors, it can eliminate them in all other private reconstructions, remaining with less than $t/2$ errors in *all* the m executions. The functionality (denoted as $\mathcal{F}_{\text{rec-g}}^{\text{batched}}$), the full specification of the protocol (denoted as $\Pi_{\text{rec-g}}^{\text{batched}}$) as well as the proof are given below.

Functionality 4.5.1: Batched Reconstruction of g -Polynomials – $\mathcal{F}_{\text{rec-g}}^{\text{batched}}$

1. **Input:** All honest parties send to the functionality the sets ZEROS $\subset [n]$ and CONFLICTS $\subset [n]$, each honest $j \notin \text{CONFLICTS}$ sends $(f_i^\ell(x), g_i^\ell(y))_{\ell \in [m]}$. Let $S_1(x, y), \dots, S_m(x, y)$ be the unique bivariate polynomials of degree at most $3t/2$ in x and at most $t + d$ in y that satisfy $f_j^\ell(x) = S^\ell(x, j)$ and $g_j^\ell(y) = S^\ell(j, y)$ for every $j \notin \text{CONFLICTS}$ and $\ell \in [m]$. Moreover, it holds that $n - |\text{CONFLICTS}| \geq 2t + 1 + t/2 + d$.
2. Send (ZEROS, CONFLICTS, $(S^\ell(x, i), S^\ell(i, y))_{i \in I, \ell \in [m]}$) to the adversary. If the dealer is corrupted, then send also $(S^\ell(x, y))_{\ell \in [m]}$ to the adversary.
3. Receive back from the adversary a message M .
4. **Output:**
 - (a) If $M = \text{discard}$ and the dealer is corrupted, then send `discard` to all parties.

- (b) If $M = (\text{detect}, \text{Bad})$ with $\text{Bad} \cap (\text{ZEROS} \cup \text{CONFLICTS}) = \emptyset$ and $|\text{Bad}| > t/2$, and in case of an honest dealer, then $\text{Bad} \subseteq I$, then send $(\text{detect}, \text{Bad})$ to all parties.
- (c) If $M = \text{proceed}$ then send:
for each $j \in \text{CONFLICTS}$ the output $(\text{proceed}, \perp, (S^\ell(j, y))_{\ell \in [m]})$
for each $j \notin \text{CONFLICTS}$ send $(\text{proceed}, (S^\ell(x, j))_{\ell \in [m]}, (S^\ell(j, y))_{\ell \in [m]})$.
-

Protocol 4.5.2: Batched Reconstruction of g -Polynomials in CONFLICTS – $\Pi_{\text{rec-g}}^{\text{batched}}$

Input: as in the functionality.

The protocol:

1. Set $\text{HAVE-SHARES} = [n] \setminus (\text{ZEROS} \cup \text{CONFLICTS})$, and $\text{localBad}_i = \emptyset$.
2. For every $j \in \text{CONFLICTS}$:
 - (a) Each party P_i for $i \in \text{HAVE-SHARES}$ sends $(i, j, (f_i^\ell(j))_{\ell \in [m]})$ to P_j .
 - (b) Let $(i, (u_i^\ell)_{\ell \in [m]})$ be the values P_j received from P_i . Moreover, for every $i \in \text{ZEROS}$, consider (i, u_i^ℓ) with $u_i^\ell = 0$. For every $\ell \in [m]$, given all $(i, u_i^\ell)_{i \notin \text{CONFLICTS}}$, P_j looks for a codeword of a polynomial of degree $t + d$ with a distance of at most $t/2$ from all the values it received (see Corollary 4.3.2, item 1). If there is such a codeword, set $g_j^\ell(y)$ to be the unique Reed Solomon reconstruction.
 - (c) If there is no such a unique codeword for some ℓ , then P_j broadcasts $\text{complaint}(j, \beta)$, where β is the minimal index in $[m]$ where reconstruction of $g_\ell(y)$ failed.
 - (d) Every party P_i for $i \in \text{HAVE-SHARES}$ broadcasts $\text{reveal}(i, j, f_i^\beta(j), \beta)$.
3. The dealer sets $\text{Bad} = \emptyset$. For each $\text{reveal}(i, j, u, \ell)$ message broadcasted, the dealer verifies that $u = f_i^\ell(j) = S^\ell(j, i)$. If not, then it adds i to Bad . The dealer broadcasts Bad .
4. Verify that (i) $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| \leq t$; and (ii) $\text{Bad} \subset \text{HAVE-SHARES}$. Otherwise, discard – go to Step 8a.
5. If $|\text{Bad}| > t/2$ then there is a large detection – go to Step 8b.
6. Otherwise, for every $j \in \text{CONFLICTS}$, if $\text{complaint}(j, \beta)$ was broadcasted, then consider all the points $R_j = \{(i, u_i^\beta)\}$ such that $\text{reveal}(i, j, u_i^\beta)$ was broadcasted in Step 2c, and $i \in \text{HAVE-SHARES} \setminus \text{Bad}$, or $u_i^\beta = 0$ if $i \in \text{ZEROS}$. Verify that R_j defines a unique polynomial of degree $t + d$. If not, discard – go to Step 8a.
7. If the dealer is not publicly discarded, then each P_j sets $g_j^\ell(y)$ to be the unique bivariate decoding of the points (i, u_i^β) as specified in the previous step. It defines the set localBad_j

as follows: $\text{localBad}_j = \{i \in [n] \mid g_j^\beta(i) \neq u_i^\beta \text{ where } u_i^\beta \text{ was received in Step 2b}\}$, i.e., it detects all the parties that provided it wrong values. Then, for every $\ell \in [m]$ it reconstructs (see Corollary 4.3.2, item 2) the unique polynomial $g_j^\ell(y)$ from the points $(u_i^\ell)_{i \notin \text{localBad}_j}$.

8. Output:

- (a) Discard the dealer: Output discard.
 - (b) Detect: Output (detect, Bad).
 - (c) Proceed: Each party $j \in \text{CONFLICTS}$ outputs (proceed, $\perp, (g_j^\ell(y))_{\ell \in [m]}$). All other parties $j \notin \text{CONFLICTS}$ output (proceed, $(f_j^\ell(x), g_j^\ell(y))_{\ell \in [m]}$).
-

Lemma 4.5.3. *Protocol $\Pi_{\text{rec-g}}^{\text{batched}}$ (Protocol 4.5.2), perfectly-securely computes $\mathcal{F}_{\text{rec-g}}^{\text{batched}}$ (Functionality 4.5.1) in the presence of a malicious adversary, controlling at most $t < n/3$.*

Proof. The analysis is a direct generalization of that of Theorem 4.4.6 and we omit the description of the simulator, where the only difference is running it multiple polynomials and not just one. We just highlight the non-trivial changes in the proof.

As previously, an honest party never belongs to Bad and we have that $\text{ZEROS} \subseteq I$, and the dealer is never discarded. If there exists an honest party that outputs (detect, Bad) then all honest party output (detect, Bad). If there exists an honest party that outputs proceed then all honest parties have their g polynomials. Specifically, consider an honest party in CONFLICTS. It receives $t + 1 + t/2 + d$ correct points on each polynomial. Since the output of this iteration is not detect, it must hold that the dealer sent a set Bad with $|\text{Bad}| < t/2$. However, we claim that $|\text{localBad}_j| \geq t/2$. If P_j did not succeed to reconstruct the β th instance, then the adversary must have introduced at least $t/2$ errors in the private points sent to P_j . Moreover, if $|\text{Bad}| < t/2$, then it must hold that corrupted parties provided public points reveal that are not the same as they sent privately to P_j . Thus, P_j must identify at least $t/2$ parties in localBad_j , and they must all be corrupted parties. Once P_j identifies $t/2$ corrupted parties, and the number of corrupted parties is bounded by t , the number of possible errors introduced in each one of the polynomials $g_\ell(y)$ for $\ell \in [m]$ is $< t/2$. P_j therefore succeeds to find unique decoding for all those polynomials.

The case of a corrupted dealer. The simulator is similar to that in the proof of Theorem 4.4.6. Again, if some honest party P_j with $j \notin \text{CONFLICTS}$ outputs (proceed, $(S^\ell(x, j))_{\ell \in [m]}, (S^\ell(j, y))_{\ell \in [m]}$) then all honest parties output proceed. Moreover, honest parties not in CONFLICTS output both f and g 's shares, and parties in CONFLICTS

output the g 's shares. This is because all parties not in CONFLICTS decide what to output according to the public view. As for the parties in CONFLICTS, we have a similar argument to that in the case of an honest dealer: once $|\text{Bad}| < t/2$ it must hold that $|\text{localBad}_j| \geq t/2$, and given that each party have $t + 1 + t/2 + d$ correct points on each polynomial, it can always recover the underlying polynomial. \square

Batched reconstruction of f -polynomials in CONFLICTS. This follows the exact same lines as the reconstruction of g polynomials. Specifically, if the local reconstruction is not unique, then it is enough to pick one instance $\gamma \in [m]$ and open it publicly. The public verification happens with respect to the γ th instance. P_j will then be able to reconstruct f_j^ℓ for every $\ell \in [m]$.

4.5.1 Sharing

To conclude, we realize the following functionality putting together the batched version of protocols for the sharing attempt, reconstruction of g and f polynomials. Referring the protocol as $\Pi_{\text{PSS}}^{\text{batched}}$, we culminate at the following theorem.

Functionality 4.5.4: Batched and Packed Secret Sharing – $\mathcal{F}_{\text{PSS}}^{\text{batched}}$

The functionality is parameterized by the set of corrupted parties $I \subseteq [n]$.

- **Input:** All parties input a set $\text{ZEROS} \subset [n]$ such that $|\text{ZEROS}| \leq t$. If the dealer is honest then it is guaranteed that $\text{ZEROS} \subseteq I$.
 - **Honest dealer:** The dealer sends $(\text{SECRETS}_\ell)_{\ell \in [m]}$ to $\mathcal{F}_{\text{PSS}}^{\text{batched}}$. The functionality sends ZEROS to the adversary, who sends back $(f_i^\ell(x), g_i^\ell(y))_{i \in I, \ell \in [m]}$ such that $f_i^\ell(k) = g_k^\ell(i)$ for every $i, k \in I$ and $\ell \in [m]$. Moreover, for every $i \in \text{ZEROS}$, $f_i(x) = g_i(y) = 0$. For every $\ell \in [m]$, the functionality chooses a random bivariate polynomial $S^\ell(x, y)$ of degree $3t/2$ in x and $t + d$ in y under the constraints that (i) SECRETS_ℓ is embedded in S_ℓ ; (ii) $S^\ell(x, i) = f_i^\ell(x)$ for every $i \in I$; (iii) $S^\ell(i, y) = g_i^\ell(y)$.
 - **Corrupted dealer:** For every $\ell \in [m]$, the dealer sends $S^\ell(x, y)$ to $\mathcal{F}_{\text{PSS}}^{\text{batched}}$ that verifies that $S^\ell(x, y)$ is of degree $3t/2$ in x and degree $t + d$ in y , and for every $i \in \text{ZEROS}$ it holds that $f_i(x) = g_i(y) = 0$. If not, $\mathcal{F}_{\text{PSS}}^{\text{batched}}$ replaces $S^\ell(x, y) = \perp$.
 - **Output:** $\mathcal{F}_{\text{PSS}}^{\text{batched}}$ sends to each party P_j the polynomials $(S^\ell(x, j), S^\ell(j, y))_{\ell \in [m]}$.
-

Theorem 4.5.5. $\Pi_{\text{PSS}}^{\text{batched}}$ securely computes $\mathcal{F}_{\text{PSS}}^{\text{batched}}$ (Functionality 4.5.4). It requires a communication complexity of $\mathcal{O}(mn^2 \log n)$ bits over-point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits

broadcast for sharing $\mathcal{O}((d+1)mn)$ values (i.e., $\mathcal{O}((d+1)mn \log n)$ bits) simultaneously in $\mathcal{O}(1)$ rounds. Each party broadcasts at most $\mathcal{O}(n \log n)$ bits.

4.5.2 Reconstruction

We present the reconstruction protocols for our batched and packed secret sharing. As mentioned in the introduction, for our detectable secret sharing, we get a detectable reconstruction, a weaker form of robust reconstruction. For the case of $d = 0$, we get robust reconstruction, and so verifiable secret sharing. We start with fully specifying the functionality.

Functionality 4.5.6: Detectable Reconstruction for Batched and Packed Secret Sharing

– $\mathcal{F}_{\text{PSS-Rec}}^{\text{batched}}$

The functionality is parameterized with the set of corrupted parties $I \subset [n]$.

1. **Input:** All honest parties send $\text{ZEROS} \subset [n]$. When the dealer is honest, $\text{ZEROS} \subseteq I$. Each honest party P_j sends $(f_j^k(x), g_j^k(y))$ for each $k \in [m]$ and $j \notin I$. For each k , let $S^k(x, y)$ be the unique bivariate polynomial of degree $3t/2$ in x and $t+d$ in y that satisfies $f_j^k(x) = S^k(x, j)$ and $g_j^k(y) = S^k(j, y)$ for every $j \notin I$.
 2. Send ZEROS and $S^1(x, y), \dots, S^m(x, y)$ to the adversary. If $d = 0$ then go to Step 4c.
 3. Receive back from the adversary a message M .
 4. **Output:**
 - (a) If $M = \text{discard}$ and the dealer is corrupted, then send discard to all parties.
 - (b) If $M = (\text{detect}, \text{Bad})$ with $|\text{Bad}| > t/2$ and $\text{Bad} \cap \text{ZEROS} = \emptyset$, and in case of an honest dealer $\text{Bad} \subseteq I$, then send $(\text{detect}, \text{Bad})$ to all parties.
 - (c) If $M = \text{proceed}$ then send to each j the output $(\text{proceed}, S^1(x, y), \dots, S^m(x, y))$.
-

Note that if the dealer is honest then discard cannot occur. Moreover, if the dealer is honest and $|\text{ZEROS}| > t/2$, the $(\text{detect}, \text{Bad})$ cannot occur, as $|\text{Bad} \cup \text{ZEROS}| \leq t$ and so we cannot have $|\text{Bad}| > t/2$. In that case, we always succeed to reconstruct. On the other hand, if the dealer is honest and $|\text{ZEROS}| \leq t/2$, the adversary might cause to a failure. In that case, we are guaranteed to have a mass detection.

The protocol. To reconstruct shared polynomials $S^1(x, y), \dots, S^m(x, y)$, the reconstruction protocol follows a similar structure of that of Protocol 4.5.2:

1. Each party P_i holds $(f_i^\ell, g_i^\ell(y))_{\ell \in [m]}$ and a set $\text{ZEROS} \subset [n]$.
2. Each party now sends all its polynomials $f_i^1(x), \dots, f_i^m(x)$ over the private channel to all other parties.

3. The parties try to reconstruct polynomials $g_1^\ell(y), \dots, g_n^\ell(y)$ using the polynomials $f_1^\ell(x), \dots, f_n^\ell(x)$ (and taking 0 for the parties in ZEROS). E.g., reconstruct $g_j^\ell(y)$ by considering $(k, f_k^\ell(j))_{k \notin \text{ZEROS}}$ and adding $(k, 0)$ for $k \in \text{ZEROS}$. Try to correct at most $t/2$ errors, for every $\ell \in [m]$ (see Corollary 4.3.4, item 1). If some party fails to decode some polynomial $g_j^\ell(y)$, then it broadcast $\text{complaint}(j, \ell)$. Note that it is enough to broadcast just a single complaint, say the one with the lexicographically smallest j, ℓ .
4. We will have a public reconstruction of $g_j^\ell(y)$: Each party broadcasts its point on that polynomial, and the dealer broadcasts a set Bad if there are any wrong values broadcasted. The parties output $(\text{detect}, \text{Bad})$ if $|\text{Bad}| > t/2$. The parties check that when excluding all points in Bad then all points lie on a single polynomial $g_j^\ell(y)$.
5. Using the public reconstruction, the party P_j can now locate $t/2$ corruptions and reconstruct (see Corollary 4.3.4, item 2) all polynomials $g_1^\ell(y), \dots, g_n^\ell(y)$ for every $\ell \in [m]$. All parties can now find unique bivariate polynomials $S^\ell(x, y)$ satisfying $S^\ell(i, y) = g_i^\ell(y)$ for every $i \in [n]$. The parties output those polynomials.

There are few properties that we would like to highlight with respect to the above protocol:

1. Note that when $d = 0$, then we can simply run Reed-Solomon decoding in Step 3 and always succeed to reconstruct as Reed Solomon decoding returns unique decoding when there are at most t errors. Thus, there is no need for public resolution.
2. There are at most n complaints, which lead to each party broadcasting at most $\mathcal{O}(n \log n)$ bits to resolve all complaints.

Conclusion: Detectable Secret Sharing. While we provide functionality-based modeling and proofs, the verifiable secret sharing literature is also full of property based definitions, and some readers might find such modeling helpful. We provide here such properties for completeness. From combining Functionalities 4.5.4 and 4.5.6, when using $d > 0$ we obtain a two-phase protocol for parties $\mathcal{P} = \{P_1, \dots, P_n\}$ where a distinguished dealer $P^* \in \mathcal{P}$ holds initial SECRETS, and all honest parties hold the same set $\text{ZEROS}_{P^*} \subseteq [n]$ (where no honest party is in ZEROS_{P^*} if P^* is honest) such that the following properties hold:

- **Secrecy:** If the dealer is honest during the first phase (the sharing phase), then at the end of this phase, the joint view of the malicious parties is independent of the dealer's input SECRETS.
- **Reconstruction or detection – corrupted dealer:** At the end of the sharing phase, the joint view of the honest parties define values $\text{SECRETS}'$ such that at the end of the

reconstruction phase – all honest parties will output either $\text{SECRETS}'$, or discard the dealer, or $t/2$ new values will be added to ZEROS_{P^*} .

- **Reconstruction or detection – honest dealer:** At the end of the sharing phase, the joint view of the honest parties define values $\text{SECRETS}' = \text{SECRETS}$ that the dealer used as input for the sharing phase. At the end of the reconstruction phase, all honest parties will output SECRETS , or $t/2$ new indices, all of corrupted parties, will be added to ZEROS_{P^*} . If ZEROS_{P^*} initially contained more than $t/2$ values during the sharing phase, then the output of the second phase is always SECRETS .

When $|\text{SECRETS}| \in \Omega(n^2)$, the protocol uses $\mathcal{O}(n^4 \log n + |\text{SECRETS}| \log n)$ communication complexity for both sharing and reconstruction.

Conclusion: Verifiable Secret Sharing. From combining Functionalities 4.5.6 and 4.5.6, when using $d = 0$ we obtain a verifiable secret sharing: A two-phase protocol for parties $\mathcal{P} = \{P_1, \dots, P_n\}$ where a distinguished dealer $P^* \in \mathcal{P}$ holds initial secrets s_1, \dots, s_t is a Verifiable Secret Sharing Protocol tolerating t malicious parties and the following conditions hold for any adversary controlling at most t parties:

- **Validity:** Each honest party P_i outputs the values $s_{i,1}, \dots, s_{i,t}$ at the end of the second phase (the reconstruction phase). Furthermore, if the dealer is honest then $(s_{i,1}, \dots, s_{i,t}) = (s_1, \dots, s_t)$.
- **Secrecy:** If the dealer is honest during the first phase (the sharing phase) then at the end of this phase, the joint view of the malicious parties is independent of the dealer's input s_1, \dots, s_t .
- **Reconstruction:** At the end of the sharing phase, the joint view of the honest parties defines values s'_1, \dots, s'_t such that all honest parties will output s'_1, \dots, s'_t at the end of the reconstruction phase.

When $|\text{SECRETS}| \in \Omega(n)$, the protocol uses $\mathcal{O}(n^4 \log n + |\text{SECRETS}| \cdot n \log n)$ communication complexity for both sharing and reconstruction.

4.6 Packed Verifiable Triple Sharing

Packed verifiable triple sharing (VTS) allows a dealer to verifiably share $t/2 + 1$ multiplication triples at the cost of incurring $\mathcal{O}(n^2)$ elements of communication over point-to-point channels as well as broadcast. Precisely, VTS outputs each element of the triples to be Shamir-shared via a degree- t polynomial. In the next section, we will present the batched version, where

$\mathcal{O}(mn)$ shared triplets are prepared with $\mathcal{O}(mn^2)$ elements of communication over point-to-point channels and the same broadcast as needed for one instance (i.e. $\mathcal{O}(n^2)$). This is an important contribution of this work that utilizes our both verifiable secret sharing and detectable secret sharing constructions.

Ideally, in a VTS, for an honest dealer, the protocol guarantees privacy of the triples, while for a corrupt dealer, it ensures correctness of the triples (i.e. they satisfy product relation) if the dealer is not discarded. However, here we construct only a weak version where it is possible for an adversary to learn the triples even when the dealer is honest. Interestingly, we show this is sufficient for our purpose, for there is a way to overcome this shortcoming in the batched version, where we show how to bound the total number of compromised instances to at most n . Furthermore, the identities of the compromised instances will be publicly known. Discarding these instances, we will still have enough ‘safe’ instances that satisfy the desired qualities, when m is sufficiently large.

4.6.1 The High-Level Idea

In order to explain VTS, we require to break open the bivariate polynomial shared through the packed secret sharing and interpret the secrets shared through univariate polynomials. Recall the way the secret-matrix SECRETS is planted in the bivariate polynomial $S(x, y)$: $S(-a, -b) = \text{SECRETS}(a, b)$ for every $a \in \{0, \dots, t/2\}$ and $b \in \{0, \dots, d\}$. At the end of PSS, every party P_i holds $f_i(x), g_i(y)$.

1. *Sharing via degree- t polynomial aka. Shamir-sharing:* First assume that $d = 0$ which is the case for the verifiable PSS: here SECRETS is a $(t/2 + 1)$ -length vector and we can treat that the ℓ th secret in SECRETS is Shamir-shared via the degree- t polynomial $g_{-\ell}(y) = S(-\ell, y)$, for $\ell = \{0, \dots, t/2\}$. Note that every party P_i holds a share on $g_{-\ell}(y)$ which is $f_i(-\ell)$.
2. *Sharing via degree- $3t/2$ polynomial:* Now assume that $d = t/4$ which is the case for the detectable PSS: here SECRETS is a $(t/2 + 1) \times (t/4 + 1)$ matrix and we can treat the degree- $3t/2$ polynomial $f_{-\ell}(x) = S(x, -\ell)$ as the packed-sharing of the ℓ th column of SECRETS, for $\ell = \{0, \dots, t/4\}$. Note that every party P_i holds a share on $f_{-\ell}(x)$ which is $g_i(-\ell)$.

We are now ready to explain the high-level idea of the packed VTS. The goal here is to generate Shamir-sharing of three vectors of secrets $\text{SECRETS}_a = \{a_0, \dots, a_{t/2}\}$, $\text{SECRETS}_b = \{b_0, \dots, b_{t/2}\}$, $\text{SECRETS}_c = \{c_0, \dots, c_{t/2}\}$, each of size $t/2 + 1$ such that $c_i = a_i b_i$ holds for

each $i \in \{0, \dots, t/2\}$. Our packed VTS consists of two phases, of which the latter is the core contribution here.

- Share degree- $(3t/2, t)$ bivariate polynomials A, B, C which embed $t/2 + 1$ multiplication triples via verifiable PSS (Functionality 4.4.10) with $d = 0$. Following this, every party P_i holds $A(x, i), A(i, y), B(x, i), B(i, y), C(x, i), C(i, y)$. The secrets for the triplets appear at $A(-\ell, 0), B(-\ell, 0)$ and $C(-\ell, 0)$ for every $\ell \in \{0, \dots, t/2\}$ and are therefore t -shared via $A(-\ell, y), B(-\ell, y)$ and $C(-\ell, y)$ (see Item 1 above).
- A proof of product relation for the embedded secrets. The dealer needs to prove that *the product of the secrets Shamir shared via $A(-\ell, y)$ and $B(-\ell, y)$ is the secret Shamir shared via $C(-\ell, y)$* . The constant term of the product polynomial $A(-\ell, y)B(-\ell, y)$ must match with the constant term of $C(-\ell, y)$. That is, the degree $2t$ polynomial $A(-\ell, y)B(-\ell, y) - C(-\ell, y)$ must have zero in its free term. This is the check we want to conduct.

Note that it is possible that this proof leaks an honest dealer's secrets, in which case the parties will kill (discard) this instance. But if it does not, then it is perfect proof in zero knowledge.

To conduct the proof, similar to [6], the dealer finds the unique bivariate polynomial $E_{-\ell}(y)$ of degree (at most) $2t$ defined by the polynomials

$$A(-\ell, y)B(-\ell, y) - C(-\ell, y)$$

for each $\ell \in \{0, \dots, t/2\}$ (see Figure 4.1a). Following this, the dealer distributes these polynomials by sharing its coefficients using the PSS (Functionality 4.4.10), with $d = t/4$. Specifically, let e_i denote the vector of the i th coefficients of all the $t/2 + 1$ $E_{-\ell}(y)$ polynomials (see Figure 4.1b). The dealer views batches of $d = t/4 + 1$ e_i s as the secrets, and invokes Functionality 4.4.10 at most eight times with $d = t/4$ (see Figure 4.1b-4.1c). This is because there are $2t + 1$ e_i s and each batch can contain $t/4 + 1$ e_i s. After these invocations of PSSs, through the f -polynomials, the parties hold a degree- $3t/2$ packed sharing of e_i (see Item 2 above). That is, parties hold $2t + 1$ packed-sharings each of degree $3t/2$ corresponding to $2t + 1$ coefficient vectors $(e_i)_{i \in \{0, \dots, 2t\}}$. Denote these polynomials by $f^{e_i}(x)$. Using linearity, these packed-sharings allow for *local* computation of degree $3t/2$ packed-sharing of evaluation points on the $t/2 + 1$ polynomials $E_{-\ell}(y)$. Let $f^{E(i)}$ denote the degree $3t/2$ polynomial which shares the evaluation of these polynomials at $y = i$ (see Figure 4.1d). That is, $f^{E(i)}$ shares the secrets $E(i) = (E_{-t/2}(i), \dots, E_0(i))$. Next assume that each P_i has access to $f^{E(i)}, f^{E(0)}$ or equivalently $E(i)$ and $E(0)$. Then, to verify the product relation, each party

$$\begin{aligned}
E_0(y) &= e_{(0,0)} & e_{(0,1)}y & \dots & e_{(0,i)}y^i & \dots & e_{(0,2t)}y^{2t} \\
E_{-1}(y) &= e_{(-1,0)} & e_{(-1,1)}y & \dots & e_{(-1,i)}y^i & \dots & e_{(-1,2t)}y^{2t} \\
&\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
E_{-t/2}(y) &= e_{(-t/2,0)} & e_{(-t/2,1)}y & \dots & e_{(-t/2,i)}y^i & \dots & e_{(-t/2,2t)}y^{2t}
\end{aligned}$$

(a) The product polynomials $E_{-\ell}(y)$ for $\ell \in \{0, \dots, t/2\}$

$e_{(0,0)}$	\dots	$e_{(0,t/4)}$	\dots	$e_{(0,i)}$	\dots	$e_{(0,2t)}$
$e_{(-1,0)}$	\dots	$e_{(-1,t/4)}$	\dots	$e_{(-1,i)}$	\dots	$e_{(-1,2t)}$
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
$e_{(-t/2,0)}$	\dots	$e_{(-t/2,t/4)}$	\dots	$e_{(-t/2,i)}$	\dots	$e_{(-t/2,2t)}$
SECRETS_1			\dots	e_i		

(b) Pictorial depiction of notation and packing of coefficients of $E_{-\ell}(y)$

$e_0 \dots e_{t/4}$	\dots	$e_{t/4+1} \dots e_{2t/4+1}$	\dots	$e_{7t/4+7} \dots e_{2t}$
SECRETS_1		SECRETS_2		SECRETS_8

(c) Packing of e_i s in $\text{SECRETS}_1, \dots, \text{SECRETS}_8$

$$\begin{aligned}
\underbrace{\begin{bmatrix} E_0(i) \\ E_{-1}(i) \\ \vdots \\ E_{-t/2}(i) \end{bmatrix}}_{E(i)} &= \underbrace{\begin{bmatrix} e_{(0,0)} & \dots & e_{(0,t/4)} \\ e_{(-1,0)} & \dots & e_{(-1,t/4)} \\ \vdots & \vdots & \vdots \\ e_{(-t/2,0)} & \dots & e_{(-t/2,t/4)} \end{bmatrix}}_{e_0} \underbrace{\begin{bmatrix} 1 \\ i \\ \vdots \\ i^{t/4} \end{bmatrix}}_{e_{t/4}} + \dots + \underbrace{\begin{bmatrix} e_{(0,7t/4+7)} & \dots & e_{(0,2t)} \\ e_{(-1,7t/4+7)} & \dots & e_{(-1,2t)} \\ \vdots & \vdots & \vdots \\ e_{(-t/2,7t/4+7)} & \dots & e_{(-t/2,2t)} \end{bmatrix}}_{e_{7t/4+7}} \underbrace{\begin{bmatrix} i^{7t/4+7} \\ i^{7t/4+8} \\ \vdots \\ i^{2t} \end{bmatrix}}_{e_{2t}} \\
&\text{SECRETS}_1 && \text{SECRETS}_8
\end{aligned}$$

(d) Pictorial depiction of computation of $E(i)$

Figure 4.1: Pictorial depiction of Packed Verifiable Triple Sharing

P_i requires to check if the relation $E_{-\ell}(y) = A(-\ell, y)B(-\ell, y) - C(-\ell, y)$ holds at $y = i$ and $y = 0$ for each $\ell \in \{0, \dots, t/2\}$. Specifically, it checks if (a) $E(0)$ is $t/2$ -length zero-vector and (b) $E(i)$ is the same as the vector $(A(-\ell, i)B(-\ell, i) - C(-\ell, i))_{i \in \{0, \dots, t/2\}}$, the latter received in the first phase. If not, it complains. If no party complains, we have that $2t + 1$ honest parties verified the product relation at distinct evaluation points on polynomials of at most degree $2t$, thus ensuring correctness of the multiplication triples. Otherwise, we must find a way to

make the shares of the complaining party on A, B, C and E polynomials public and verify the product relation publicly.

The remaining tasks are (a) complaint resolution and (b) make sure P_i gets access to $E(0), E(i)$. We note that both these require reconstruction of $3t/2$ -degree polynomials: $A(x, i), B(x, i), C(x, i), f^{E(i)}$ s for the former and $f^{E(i)}$ s for the latter. The former is a public reconstruction, while the latter is private to every P_i . We therefore design a private and a public reconstruction protocol. Due to the high degree of $3t/2$, robust reconstruction via Reed-Solomon decoding is not an option. Instead, we design protocols which ensure one of the following outcomes:

- discard the dealer when it is corrupt
- identify a large of conflicts with the dealer (and restart)
- proceed with the successful reconstruction.

For the case of private reconstruction, we get a weaker guarantee wherein apart from the above three outcomes, there is a fourth possibility that the polynomials to be reconstructed privately to the honest parties are compromised. It is due to this breach, we have the weaker privacy guarantee for our VTS, namely an honest dealer's triplets may be leaked to the adversary.

Lastly, similar to PSS, here too we may have a constant number of restarts of the VTS (precisely 3) before a successful run, where the restarts can happen inside the PSS instances or as a part of the reconstructions.

Below we discuss our reconstruction protocols, and then move on to the packed VTS protocol.

4.6.2 Reconstructions

We present two variants of reconstructions— private and public.

4.6.2.1 Weak Private Reconstruction

In a private reconstruction functionality, the parties give their respective shares of $n + 1$ degree- $3t/2$ polynomials. The functionality reconstructs these polynomials and sends the 0th and the i th polynomial privately to P_i . As mentioned earlier, we have four possible outcomes here: discard the dealer, detect large number of conflicts, kill the instance (in which case the honest parties' polynomials are compromised) and proceed to successful completion.

Functionality 4.6.1: Weak Private Reconstruction of $3t/2$ -Shared Polynomials – $\mathcal{F}_{\text{privRec}}$

- **Input:** All honest parties send $\text{ZEROS} \subset [n]$. When the dealer is honest, $\text{ZEROS} \subseteq I$. Each honest party P_j inputs shares $(h_0(j), \dots, h_n(j))$ on polynomials, each of degree $3t/2$ to $\mathcal{F}_{\text{privRec}}$. The functionality reconstructs the polynomials as $h_0(x), \dots, h_n(x)$.
- **The functionality:**
 1. If the dealer is honest, then send $\text{ZEROS}, (h_0(i), \dots, h_n(i))_{i \in I}$ and $(h_0(x), h_i(x))_{i \in I}$ to the adversary. If the dealer is corrupted, then send ZEROS and $h_0(x), \dots, h_n(x)$ to the adversary.
 2. Receive a bit leak from the adversary. If $\text{leak} = 1$ and the dealer is honest then send $h_0(x), \dots, h_n(x)$ to the adversary.
 3. Receive a message M from the adversary.
 4. **Output:**
 - (a) If $M = \text{discard}$ and the dealer is corrupted, then send discard to all parties.
 - (b) If $M = (\text{detect}, \text{Bad})$ with $\text{Bad} \cap \text{ZEROS} = \emptyset$ and $|\text{Bad}| > t/2$, and in case of an honest dealer then $\text{Bad} \subseteq I$, then send $(\text{detect}, \text{Bad})$ to all parties.
 - (c) If $M = \text{kill}$ and $\text{leak} = 1$ then send kill to the parties.
 - (d) If $M = \text{proceed}$ and $\text{leak} = 0$ then send $(\text{proceed}, h_0(x), h_\ell(x))$ to each party P_ℓ .

Protocol 4.6.2: Weak Private Reconstruction of $3t/2$ -Shared Polynomials – Π_{privRec}

- **Input:** All parties hold the same set ZEROS . Each honest party P_j holds shares $(h_0(j), \dots, h_n(j))$ on n polynomials $h_0(x), \dots, h_n(x)$ each of degree $3t/2$. It is guaranteed that all the shares of honest parties lie on the same $3t/2$ degree polynomials.
- **The protocol:**
 1. Each P_j sends $(j, h_0(j), h_\ell(j))$ to every P_ℓ .
 2. Let (j, u_j, v_j) be the value P_ℓ received from P_j . For every $j \in \text{ZEROS}$, (j, u_j, v_j) is such that $u_j = v_j = 0$. Given all (j, u_j) and (j, v_j) , P_ℓ looks for a codeword of distance at most $t/2$ from all the values it received (see Corollary 4.3.3, item 1). If there is such a codeword, set $h_0(x)$ and $h_\ell(x)$ to be the unique Reed-Solomon reconstruction respectively. If there is no such a unique codeword, then P_ℓ broadcasts $\text{complaint}(\ell)$ and every party P_j broadcasts $\text{reveal}(\ell, j, h_0(j), h_\ell(j))$.
 3. If no party broadcasts $\text{complaint}(\ell)$ then go to Step 8d.
 4. The dealer sets $\text{Bad} = \emptyset$. For each $\text{reveal}(\ell, j, u, v)$ message broadcasted, the dealer verifies that $u = h_0(j)$ and $v = h_\ell(j)$. If not, then it adds j to Bad . The dealer

broadcasts Bad.

5. Verify that (i) $|\text{ZEROS} \cup \text{Bad}| \leq t$; and (ii) $\text{Bad} \subset [n] \setminus \text{ZEROS}$. Otherwise, discard the dealer and go to Step 8a.
 6. If $|\text{Bad}| > t/2$ then there is a large detection and so go to Step 8b.
 7. Otherwise, consider all the points $R_\ell = \{(j, u_j)\}$ and $T_\ell = \{(j, v_j)\}$ such that $\text{reveal}(\ell, j, u_j, v_j)$ was broadcasted in Step 2 for $j \notin \text{Bad}$ where $u_j = v_j = 0$ if $j \in \text{ZEROS}$. Verify that each R_ℓ and T_ℓ define a unique polynomial of degree $3t/2$. If not, go to Step 8a. Otherwise, go to Step 8c.
 8. **Output:**
 - (a) Discard the dealer: Output discard.
 - (b) Detect: Output (detect, Bad).
 - (c) Kill: Output kill.
 - (d) Proceed: Each P_ℓ outputs (proceed, $h_0(x), h_\ell(x)$).
-

Lemma 4.6.3. Protocol 4.6.2, Π_{privRec} perfectly securely computes Functionality 4.6.1, $\mathcal{F}_{\text{privRec}}$ in the presence of a malicious adversary controlling at most $t < n/3$.

Proof. We show the case of an honest dealer and a corrupted dealer separately.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z .
2. Receive from the functionality the sets ZEROS, $\{(h_0(i), \dots, h_n(i))\}_{i \in I}$ and $(h_0(x), h_i(x))_{i \in I}$.
3. For each $i \in I$, simulate every honest P_j sending $(j, h_0(j), h_i(j))$ to P_i . Receive $(i, h_0(i), h_j(i))$ from the adversary for each $i \in I$ and every honest P_j .
4. If for some honest P_j , (i, u_i, v_i) sent by the adversary is such that $u_i \neq h_0(i)$ or $v_i \neq h_j(i)$ for more than $t/2$ such $i \in I$ then simulate P_j broadcasting $\text{complaint}(j)$. Also listen to all $\text{complaint}(i)$ messages broadcasted by the adversary. If some party broadcasts complaint, then send $\text{leak} = 1$ to the functionality. Receive $h_0(x), \dots, h_n(x)$ from the functionality. Set $\text{Bad} = \emptyset$.
5. For each $\text{complaint}(\ell)$ broadcasted by some P_ℓ , simulate broadcasting $(\ell, j, h_0(j), h_\ell(j))$ for every honest P_j and listen to all the adversary's broadcasts.
6. For each $i \in I$ where (ℓ, i, u_i, v_i) broadcasted is such that $u_i \neq h_0(i)$ or $v_i \neq h_\ell(i)$, add i to Bad.

7. Simulate the dealer broadcasting `Bad`. If $|\text{Bad}| > t/2$ then send `(detect, Bad)` to the functionality and halt.
8. If `leak = 1` then send `kill` to the functionality and halt.
9. Otherwise, send `proceed` to the functionality and halt.

The protocol as well as the simulator is deterministic. Hence, it can be easily observed that the view of the adversary in the real and ideal executions is identical. It remains to show that the output of honest parties is identical in the real and ideal world.

Towards that end, note that in the real world, an honest dealer is discarded if and only if one of the following conditions holds:

1. $|\text{ZEROS} \cup \text{Bad}| > t$.
2. $\text{Bad} \not\subseteq [n] \setminus \text{ZEROS}$.
3. R_ℓ or T_ℓ do not define a unique polynomial of degree $3t/2$.

Note that for an honest dealer, an honest party never belongs to `Bad` and we have that $\text{ZEROS} \subseteq I$. It is clear that none of the above conditions hold and the dealer is not discarded. We thus have the following cases to consider:

1. **There exists an honest party that outputs `(detect, Bad)`:** In this case, it must hold that $|\text{Bad}| > t/2$. Since the corresponding message was broadcasted by the dealer, it must hold that all the honest parties output `(detect, Bad)`. The simulator emulates the interaction of the honest parties with the dealer as in the real execution, hence all the simulated honest parties hold the same set `Bad`. In that case, the simulator sends `(detect, Bad)` to the functionality, causing all the honest parties in the ideal world to output the same.
2. **There exists an honest party that outputs `kill`:** In this case, it must hold that $|\text{Bad}| \leq t/2$. Moreover, there exists some party P_ℓ which complained and each R_ℓ and T_ℓ define a unique degree $3t/2$ polynomials. Since all the corresponding messages are broadcast, all the honest parties would output `kill`. The simulator emulates the honest parties as in the real execution, hence all the simulated honest parties hold the same set `Bad` and see the same complaints. In this case, the simulator sends `leak = 1` to the functionality, followed by `kill`, causing all the honest parties in the ideal execution to output `kill`.
3. **There exists an honest party `proceed`:** This implies that no party broadcast complaint in the real execution. Thus, it must hold that each honest party P_ℓ obtained a unique reconstruction in Step 2, which agrees with the shares of all the honest parties. Since the reconstructed polynomials and $h_0(x), h_\ell(x)$ defined by the honest parties' input shares

are each of degree $3t/2$ and agree in at least $2t + 1$ points, it must hold that the unique reconstructed polynomials are $h_0(x), h_\ell(x)$. Hence it must hold that all honest parties output $(\text{proceed}, h_0(x), h_\ell(x))$ in the real execution. Since the simulated execution is identical to the real execution, all the simulated honest parties also see that no complaint was broadcast. In this case, the simulator sends `proceed` to the functionality, causing all the honest parties in the ideal world to have an output that is identical to the output of the honest parties in the real world.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z .
2. Receive from the functionality the set ZEROS, and the polynomials $h_0(x), \dots, h_n(x)$.
3. Simulate the protocol where each honest party P_j holds $h_0(j), \dots, h_n(j)$ and all parties have the same set ZEROS.
4. If some party broadcasts `complaint` in Step 2 then the simulator sends `leak = 1` to the functionality.
5. Send the message M to the functionality according to the following cases (the proof will show that the cases are mutually-exclusive):
 - (a) If the output of some simulated honest party is `discard`, then send `discard` to the functionality and halt.
 - (b) If the output of some simulated honest party is `(detect, Bad)`, then send `(detect, Bad)` to the functionality and halt.
 - (c) If the output of some simulated honest party is `kill` then send `kill` to the functionality and halt.
 - (d) If the output of some simulated honest party is `proceed`, then send `proceed` to the functionality and halt.

Since the simulator uses the exact same inputs of the simulated honest parties as the real honest parties in the real execution, and since the protocol is deterministic, we get that the view of the adversary is exactly the same in the real and in the ideal executions. Thus it remains to be shown that the output of the honest parties is identical in the real and ideal executions. We have the following cases to consider:

1. **There exists an honest party that outputs `discard` in the real world:** Observe that an honest party outputs `discard` if and only if one of the following conditions holds.

- (a) The dealer broadcasted Bad such that either (i) $|\text{ZEROS} \cup \text{Bad}| > t$; or (ii) $\text{Bad} \not\subset [n] \setminus \text{ZEROS}$. Since all honest parties hold the same set ZEROS , and the set Bad is broadcasted, we get that all honest parties output `discard`.
- (b) The dealer broadcasted Bad with $|\text{Bad}| \leq t/2$, and for which $|\text{ZEROS} \cup \text{Bad}| \leq t$ and $\text{Bad} \subset [n] \setminus \text{ZEROS}$. However, for $R_\ell = \{(j, u_j)\}$ and $T_\ell = \{(j, v_j)\}$ such that $\text{reveal}(\ell, j, u_j, v_j)$ was broadcasted in Step 2 and $j \notin \text{Bad}$, it holds that R_ℓ or T_ℓ does not define a unique polynomial of degree $3t/2$. Since the set R_ℓ and T_ℓ are public, all honest parties will identify that there is no unique reconstruction, and all would output `discard`.

Since the simulated honest parties have the same view as the honest parties, the simulated honest parties also output `discard`. In this case, the simulator sends `discard` to the functionality causing all the honest parties to output `discard` in the ideal execution.

2. **There exists an honest party that outputs `(detect, Bad)` in the real world:** In this case, it must hold that $|\text{ZEROS} \cup \text{Bad}| \leq t$ and $\text{Bad} \subset [n] \setminus \text{ZEROS}$. Moreover, it must hold that $|\text{Bad}| \geq t/2$. Since the corresponding set Bad is broadcast, all the honest parties hold the same set and hence output `(detect, Bad)`. The simulated honest parties hold an identical output, and thus the simulator sends `(detect, Bad)` to the functionality, which in turn sends the same to all the honest parties in the ideal execution.
3. **There exists an honest party that outputs `kill` in the real world:** In this case, it must hold that some party P_ℓ broadcast `complaint`(ℓ). Moreover, $|\text{Bad}| \leq t/2$ and R_ℓ and T_ℓ each define a unique degree $3t/2$ polynomials. Since all the corresponding messages are broadcast, it must hold that all the honest parties output `kill`. In this case, the simulated honest parties also hold an identical output. Since the simulated honest parties also observe the `complaint`(ℓ) broadcast by P_ℓ , the simulator sends `leak = 1` to the functionality, followed by $M = \text{kill}$ causing all the honest parties in the ideal world to output `kill`.
4. **There exists an honest party that outputs `proceed` in the real world:** In this case, we show that all honest parties P_ℓ output `(proceed, $h_0(x), h_\ell(x)$)` where $h_0(x), h_\ell(x)$ are the degree $3t/2$ polynomials that are interpolated from the respective input shares of the honest parties and are guaranteed to exist under our input assumption. Towards that, observe that since there exists an honest party that does not output `discard`, `(detect, Bad)` or `kill` it must hold that no party broadcasted `complaint` in Step 2. Hence, it must hold that each honest P_ℓ has a unique reconstruction and consequently lesser than $t/2$ errors occurred in the reconstruction for P_ℓ . The reconstructed polynomials

agree with the shares of at least $n - t/2 \geq 5t/2 + 1$ parties, hence they agree with the shares of at least $3t/2 + 1$ honest parties. Since the reconstructed polynomials and each $h_0(x), h_\ell(x)$ are of degree $3t/2$, it must hold that the reconstructed polynomials for each honest P_ℓ are $h_0(x), h_\ell(x)$ consistent with the input shares of the honest parties. In this case, the simulated honest parties also observe that no party complains and hence the simulator sends proceed to the functionality, causing all the honest parties in the ideal world to obtain an output identical to the output in the real execution. □

Lemma 4.6.4. *Let $t < n/3$. There exists a protocol that implements, Functionality 4.6.1, $\mathcal{F}_{\text{privRec}}$, has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast in $\mathcal{O}(1)$ rounds. Every party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

4.6.2.2 Public Reconstruction

Similar to private reconstruction, shares of n degree- $3t/2$ polynomials are taken as the input by the functionality. It also receives n flags where i th one indicates if the i th polynomial needs to be made public.

Functionality 4.6.5: Public Reconstruction of $3t/2$ -Shared Polynomials – $\mathcal{F}_{\text{pubRec}}$

- **Input:** All honest parties send $\text{ZEROS} \subset [n]$ and a binary vector $(\text{pub}_1, \dots, \text{pub}_n)$. When the dealer is honest, $\text{ZEROS} \subseteq I$. Each honest party P_j inputs shares $(h_1(j), \dots, h_n(j))$ on n polynomials, each of degree $3t/2$ to $\mathcal{F}_{\text{pubRec}}$. The functionality reconstructs the polynomials as $h_1(x), \dots, h_n(x)$.
- **The functionality:**
 1. If the dealer is honest, then send $\text{ZEROS}, (\text{pub}_1, \dots, \text{pub}_n), h_\ell(i)$ for each $i \in I$ and every $\ell \in [n], h_i(x)$ for every $i \in I$ and $h_\ell(x)$ for every $\ell \in [n]$ such that $\text{pub}_\ell = 1$ to the adversary. If the dealer is corrupted, then send $\text{ZEROS}, (\text{pub}_1, \dots, \text{pub}_n)$ and $h_1(x), \dots, h_n(x)$ to the adversary.
 2. Receive a message M from the adversary.
 3. **Output:**
 - (a) If $M = \text{discard}$ and the dealer is corrupted, then send discard to all parties.
 - (b) If $M = (\text{detect}, \text{Bad})$ with $\text{Bad} \cap \text{ZEROS} = \emptyset$ and $|\text{Bad}| > t/2$, and in case of an honest dealer then $\text{Bad} \subseteq I$, then send $(\text{detect}, \text{Bad})$ to all parties.
 - (c) If $M = \text{proceed}$ then send $(\text{proceed}, \{h_\ell(x)\}_{\ell \text{ s.t. } \text{pub}_\ell=1})$ to all parties.

Protocol 4.6.6: Public Reconstruction of $3t/2$ -Shared Polynomials – Π_{pubRec}

- **Input:** All parties hold the same set ZEROS. Each honest party P_j holds shares $(h_1(j), \dots, h_n(j))$ on n polynomials $h_1(x), \dots, h_n(x)$ each of degree $3t/2$. It is guaranteed that all the shares of honest parties lie on the same $3t/2$ degree polynomials. All parties hold the same binary vector $(\text{pub}_1, \dots, \text{pub}_n)$.
 - **The protocol:**
 1. Each P_j broadcasts $(\ell, j, h_\ell(j))$ for every ℓ such that $\text{pub}_\ell = 1$.
 2. Let $(\ell, j, u_{\ell,j})$ be the value P_j broadcasted. For every $j \in \text{ZEROS}$, $(\ell, j, u_{\ell,j})$ is such that $u_{\ell,j} = 0$.
 3. The dealer sets $\text{Bad} = \emptyset$. For each $(\ell, j, u_{\ell,j})$ message broadcasted, the dealer verifies that $u_{\ell,j} = h_\ell(j)$. If not, then it adds j to Bad . The dealer broadcasts Bad .
 4. Verify that (i) $|\text{ZEROS} \cup \text{Bad}| \leq t$; and (ii) $\text{Bad} \subset [n] \setminus \text{ZEROS}$. Otherwise, discard the dealer and go to Step 7a.
 5. If $|\text{Bad}| > t/2$ then there is a large detection and so go to Step 7b.
 6. Otherwise, consider all the points $R_\ell = \{(j, u_{\ell,j})\}$ such that $(\ell, j, u_{\ell,j})$ was broadcasted in Step 1 for $j \notin \text{Bad}$ where $u_{\ell,j} = 0$ if $j \in \text{ZEROS}$. Verify that each R_ℓ defines a unique polynomial of degree $3t/2$. If not, go to Step 7a. Otherwise, set $h_\ell(x)$ to be this unique polynomial and go to Step 7c.
 7. **Output:**
 - (a) Discard the dealer: Output `discard`.
 - (b) Detect: Output `(detect, Bad)`.
 - (c) Proceed: All parties output `(proceed, $\{h_\ell(x)\}_{\ell \text{ s.t. } \text{pub}_\ell=1}$)`.
-

Lemma 4.6.7. Protocol 4.6.6, Π_{pubRec} , perfectly securely computes Functionality 4.6.5, $\mathcal{F}_{\text{pubRec}}$ in the presence of a malicious adversary controlling at most $t < n/3$.

Proof. We show the case of an honest dealer and a corrupted dealer separately.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z .

2. Receive from the functionality the sets ZEROS, $(\text{pub}_1, \dots, \text{pub}_n)$ $\{h_\ell(i)\}_{i \in I}$ and $h_\ell(x)$ for each ℓ such that $\ell \in I$ or $\text{pub}_\ell = 1$.
3. For each ℓ such that $\text{pub}_\ell = 1$, simulate every honest P_j broadcasting $(\ell, j, h_\ell(j))$. Listen to the broadcasts $(\ell, i, h_\ell(i))$ from the adversary for each $i \in I$. Set $\text{Bad} = \phi$.
4. For each $(\ell, i, u_{\ell,i})$ broadcasted by the adversary, if $u_{\ell,i} \neq h_\ell(i)$ then add i to Bad.
5. Simulate the dealer broadcasting Bad. If $|\text{Bad}| > t/2$ then send $(\text{detect}, \text{Bad})$ to the functionality and halt.
6. Otherwise, send proceed to the functionality and halt.

The protocol as well as the simulator is deterministic. Hence, it can be easily observed that the view of the adversary in the real and ideal executions is identical. Thus, it remains to be shown that the output of honest parties is identical in the real and ideal world.

Towards that end, note that in the real world, an honest dealer is discarded if and only if one of the following conditions holds:

1. $|\text{ZEROS} \cup \text{Bad}| > t$.
2. $\text{Bad} \not\subseteq [n] \setminus \text{ZEROS}$.
3. R_ℓ does not define a unique polynomial of degree- $3t/2$.

Note that for an honest dealer, an honest party never belongs to Bad and we have that $\text{ZEROS} \subseteq I$. It is clear that none of the above conditions hold and the dealer is not discarded. We thus have the following two cases to consider:

1. **There exists an honest party that outputs $(\text{detect}, \text{Bad})$:** In this case, it must hold that $|\text{Bad}| > t/2$. Since the corresponding message was broadcasted by the dealer, it must hold that all the honest parties output $(\text{detect}, \text{Bad})$. The simulator emulates the the honest parties as in the real execution, hence all the simulated honest parties hold the same set Bad and the same output. In this case, the simulator sends $(\text{detect}, \text{Bad})$ to the functionality, causing all the honest parties in the ideal world to output the same.
2. **There exists an honest party proceed:** In this case, we show that all the honest parties output $(\text{proceed}, \{h_\ell(x)\}_{\ell \text{ s.t. } \text{pub}_\ell=1})$. For this, note that since there exists an honest party that does not output discard or $(\text{detect}, \text{Bad})$, it must hold that $|\text{ZEROS} \cup \text{Bad}| \leq t$, $\text{Bad} \subset [n] \setminus \text{ZEROS}$ and $|\text{Bad}| \leq t/2$. Moreover, for every ℓ with $\text{pub}_\ell = 1$, the points R_ℓ which were broadcasted (excluding the points of parties in Bad) define a unique degree- $3t/2$ polynomial $h_\ell(x)$. Since all the corresponding messages were broadcast, all the honest parties have the same view and hence output $(\text{proceed}, \{h_\ell(x)\}_{\ell \text{ s.t. } \text{pub}_\ell=1})$. Since the real and simulated executions are identical, the simulated honest parties have

the same output. In this case, the simulator sends proceed to the functionality, causing the honest parties in the ideal world to receive an identical output.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z .
2. Receive from the functionality the set ZEROS, $(\text{pub}_1, \dots, \text{pub}_n)$ and the polynomials $h_1(x), \dots, h_n(x)$.
3. Simulate the protocol where each honest party P_j holds $h_1(j), \dots, h_n(j)$ and all parties have the same set ZEROS and $(\text{pub}_1, \dots, \text{pub}_n)$.
4. Send the message M to the functionality according to the following cases (the proof will show that the cases are mutually-exclusive):
 - (a) If the output of some simulated honest party is discard, then send discard to the functionality and halt.
 - (b) If the output of some simulated honest party is $(\text{detect}, \text{Bad})$, then send $(\text{detect}, \text{Bad})$ to the functionality and halt.
 - (c) If the output of some simulated honest party is proceed, then send proceed to the functionality and halt.

Since the simulator uses the exact same inputs of the simulated honest parties as the real honest parties in the real execution, and since the protocol is deterministic, we get that the view of the adversary is exactly the same in the real and in the ideal executions. Thus it remains to be shown that the output of the honest parties is identical in the real and ideal executions. We have the following cases to consider:

1. **There exists an honest party that outputs discard in the real world:** Observe that an honest party outputs discard if and only if one of the following conditions holds.
 - (a) The dealer broadcasted Bad such that either (i) $|\text{ZEROS} \cup \text{Bad}| > t$; or (ii) $\text{Bad} \not\subseteq [n] \setminus \text{ZEROS}$. Since all honest parties hold the same set ZEROS, and the set Bad is broadcasted, we get that all honest parties output discard.
 - (b) The dealer broadcasted Bad with $|\text{Bad}| \leq t/2$, and for which $|\text{ZEROS} \cup \text{Bad}| \leq t$ and $\text{Bad} \subseteq [n] \setminus \text{ZEROS}$. However, for $R_\ell = \{(j, u_j)\}$ such that $(\ell, j, u_{\ell,j})$ was broadcasted in Step 1 and $j \notin \text{Bad}$, it holds that R_ℓ does not define a unique polynomial of degree $3t/2$. Since the set R_ℓ is public, all honest parties will identify that there is no unique reconstruction, and all would output discard.

Since the simulated honest parties have the same view as the honest parties in the real execution, the simulated honest parties also output `discard`. In this case, the simulator sends `discard` to the functionality causing all the honest parties to output `discard` in the ideal execution.

2. **There exists an honest party that outputs `(detect, Bad)` in the real world:** In this case, it must hold that $|\text{ZEROS} \cup \text{Bad}| \leq t$ and $\text{Bad} \subset [n] \setminus \text{ZEROS}$. Moreover, it must hold that $|\text{Bad}| \geq t/2$. Since the corresponding set `Bad` is broadcast, all the honest parties hold the same set and hence output `(detect, Bad)`. The simulated honest parties hold an identical output, and thus the simulator sends `(detect, Bad)` to the functionality, which in turn sends the same to all the honest parties in the ideal execution.
3. **There exists an honest party that outputs `proceed` in the real world:** In this case, we show that all honest parties P_ℓ output `(proceed, $\{h_\ell(x)\}_{\ell \text{ s.t. } \text{pub}_\ell=1}$)` where each $h_\ell(x)$ is the degree $3t/2$ polynomial that is interpolated from the respective input shares of the honest parties and is guaranteed to exist under our input assumption. Towards that, observe that since there exists an honest party that does not output `discard`, `(detect, Bad)` it must hold that $|\text{ZEROS} \cup \text{Bad}| \leq t$, $\text{Bad} \subset [n] \setminus \text{ZEROS}$ and $|\text{Bad}| \leq t/2$. Moreover, for each $R_\ell = \{(j, u_j)\}$ such that $(\ell, j, u_{\ell,j})$ was broadcasted in Step 1 and $j \notin \text{Bad}$, it holds that R_ℓ defines a unique polynomial of degree $3t/2$. Since $|\text{Bad}| \leq t/2$, it must thus hold that the reconstructed polynomial for each ℓ agrees with shares of at least $n - t/2 \geq 5t/2 + 1$ parties. Consequently, the reconstructed polynomial agrees with shares of at least $3t/2 + 1$ honest parties. Since both the reconstructed polynomial and $h_\ell(x)$ are of degree $3t/2$, it holds that polynomial obtained from R_ℓ is indeed $h_\ell(x)$ that is defined by the input shares of the honest parties. In this case, the simulated honest parties also hold the same output and hence the simulator sends `proceed` to the functionality. Thus the output of honest parties in the ideal execution is identical to their output in the real world.

□

Lemma 4.6.8. *Let $t < n/3$. There exists a protocol that implements Functionality 4.6.5, $\mathcal{F}_{\text{pubRec}}$ and has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits broadcast in $\mathcal{O}(1)$ rounds. Every party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

4.6.3 Putting Everything Together: Packed VTS

We now present the functionality and the protocol for a packed VTS.

Functionality 4.6.9: Packed Verifiable Triple Sharing – $\mathcal{F}_{\text{PVTS}}$

The functionality is parameterized by a set of corrupted parties $I \subset [n]$.

1. Honest dealer:

- (a) The dealer sends $\text{SECRETS}_a, \text{SECRETS}_b, \text{SECRETS}_c$ to $\mathcal{F}_{\text{PVTs}}$.
- (b) The adversary sends $(f_i^a(x), g_i^a(y))_{i \in I}, (f_i^b(x), g_i^b(y))_{i \in I}, (f_i^c(x), g_i^c(y))_{i \in I}$ to $\mathcal{F}_{\text{PVTs}}$ such that $f_i^a(k) = g_k^a(i), f_i^b(k) = g_k^b(i)$ and $f_i^c(k) = g_k^c(i)$ for every $i, k \in I$.
- (c) The functionality chooses random bivariate polynomials $A(x, y), B(x, y)$ and $C(x, y)$ of degree $3t/2$ in x and t in y under the constraints that (i) $\text{SECRETS}_a, \text{SECRETS}_b, \text{SECRETS}_c$ are embedded in A, B, C respectively; (ii) $A(x, i) = f_i^a(x), B(x, i) = f_i^b(x)$ and $C(x, i) = f_i^c(x)$ for every $i \in I$; (iii) $A(i, y) = g_i^a(y), B(i, y) = g_i^b(y)$ and $C(i, y) = g_i^c(y)$ for every $i \in I$.

- 2. Corrupted dealer:** The dealer sends $A(x, y), B(x, y)$ and $C(x, y)$ to $\mathcal{F}_{\text{PVTs}}$ that verifies that (i) $A(x, y), B(x, y)$ and $C(x, y)$ are of degree $3t/2$ in x and degree t in y ; and (ii) $A(-\ell, 0) \cdot B(-\ell, 0) = C(-\ell, 0)$ holds for each $\ell \in \{0, \dots, t/2\}$. If not, $\mathcal{F}_{\text{PVTs}}$ replaces each $A(x, y), B(x, y)$ and $C(x, y)$ with \perp .

- 3.** The functionality receives from the adversary a message M .

4. Output:

- (a) If $M = \text{kill}$, then send kill to all parties and if the dealer is honest, then send $A(x, y), B(x, y), C(x, y)$ to the adversary.
- (b) Otherwise, send to each party P_j the pairs of polynomials $A(x, j), A(j, y), B(x, j), B(j, y)$ and $C(x, j), C(j, y)$.

Protocol 4.6.10: Packed VTS in the $(\mathcal{F}_{\text{PSS}}, \mathcal{F}_{\text{privRec}}, \mathcal{F}_{\text{pubRec}})$ -Hybrid model – Π_{PVTs}

Input: The dealer holds three lists $\text{SECRETS}_a = \{a_0, \dots, a_{t/2}\}, \text{SECRETS}_b = \{b_0, \dots, b_{t/2}\}, \text{SECRETS}_c = \{c_0, \dots, c_{t/2}\}$, each of size $t/2 + 1$ such that $c_i = a_i b_i$ holds for each $i \in \{0, \dots, t/2\}$.

The protocol:

1. All parties set $\text{ZEROS} = \emptyset$.
2. **Dealing the shares of triples:** Parties invoke \mathcal{F}_{PSS} (Functionality 4.4.10) three times with $d = 0$, where the dealer inputs $\text{SECRETS}_a, \text{SECRETS}_b, \text{SECRETS}_c$ respectively and each party inputs ZEROS . If the output of any instance is \perp , then proceed to Step 5a. Otherwise, each P_i holds $f_i^a(x) = A(x, i), g_i^a(y) = A(i, y), f_i^b(x) = B(x, i), g_i^b(y) = B(i, y)$ and $f_i^c(x) = C(x, i), g_i^c(y) = C(i, y)$.

3. Dealing the shares of product polynomials:

- (a) For each $\ell \in \{0, \dots, t/2\}$, the dealer defines the polynomials $E_{-\ell}(y)$ of degree at most $2t$ such that $E_{-\ell}(y) = A(-\ell, y) \cdot B(-\ell, y) - C(-\ell, y) = e_{(-\ell,0)} + e_{(-\ell,1)}y + \dots + e_{(-\ell,2t)}y^{2t}$. Define $e_i = (e_{(-t/2,i)}, \dots, e_{(0,i)})$ for $i \in \{0, \dots, 2t\}$, as the vector of i th coefficients of all the $t/2 + 1$ polynomials.
- (b) The dealer views the coefficients of these $t/2 + 1$ polynomials as (at most) eight matrices $\text{SECRETS}_1, \dots, \text{SECRETS}_8$, each of size $(t/2 + 1)(t/4 + 1)$. Specifically, $\text{SECRETS}_u(\cdot, b) = e_{(t/4+1) \cdot (u-1) + b}$ where $b \in \{0, \dots, t/4\}$, $u \in [8]$.
- (c) All the parties invoke \mathcal{F}_{PSS} (Functionality 4.4.10) (at most) eight times with $d = t/4$, where the dealer inputs SECRETS_u for each $u \in [8]$ respectively and each party inputs ZEROS. If the output of any instance is \perp , then proceed to Step 5a. Otherwise, each P_i holds the degree- $(t + t/2)$ polynomials $f_i^u(x) = E_u(x, i)$ and degree- $(t + t/4)$ polynomials $g_i^u(y) = E_u(i, y)$ for all $u \in [8]$.
- (d) Let $f^{e_{(t/4+1) \cdot (u-1) + b}}(x) = E_u(x, -b)$ for $b \in \{0, \dots, t/4\}$, $u \in [8]$, denote the $3t/2$ -degree polynomial that packed-shares the coefficient vector $e_{(t/4+1) \cdot (u-1) + b}$.

4. **Proof of product relation:** To check the product relation, each P_i requires to verify that $E_{-\ell}(i) = A(-\ell, i) \cdot B(-\ell, i) - C(-\ell, i)$ holds for each $\ell \in \{0, \dots, t/2\}$. Let $E(i) = (E_{-t/2}(i), \dots, E_0(i))$ and $f^{E(i)}(x) = \sum_{k=0}^{2t} i^k f^{e_k}(x)$ be the $3t/2$ -degree polynomial that packed-shares $E(i)$. Note that each $f^{E(i)}(x)$ is a linear combination of the polynomials $(f^{e_j}(x))_{j \in \{0, \dots, 2t\}}$.

- (a) **Reconstructing $E(i)$ and $E(0)$ towards each P_i :** For each party P_i , every party P_j computes $f^{E(i)}(j) = \sum_{k=0}^{2t} i^k \cdot f^{e_k}(j)$ and $f^{E(0)}(j) = f^{e_0}(j)$. Parties invoke $\mathcal{F}_{\text{privRec}}$ (Functionality 4.6.1) where P_j inputs $(f^{E(0)}(j), f^{E(1)}(j), \dots, f^{E(n)}(j))$ and ZEROS.
 - i. If the output is discard, then proceed to Step 5a.
 - ii. If the output is (detect, Bad), then set $\text{ZEROS} = \text{ZEROS} \cup \text{Bad}$. If $|\text{ZEROS}| > t$ then go to Step 5a. Otherwise, go to Step 2.
 - iii. If the output is kill then go to Step 5b.
 - iv. Otherwise, P_i sets $f^{E(0)}(x) = h_0(x)$ and $f^{E(i)}(x) = h_i(x)$ where $(\text{proceed}, h_0(x), h_i(x))$ is the output of $\mathcal{F}_{\text{privRec}}$.
- (b) **Verifying the product relation of each P_i and that $E(0) = e_0 = (0, \dots, 0)$ holds:**
 - i. Each P_i verifies that $E(i)$ obtained from reconstructed polynomial $f^{E(i)}(x)$ matches with $(f_i^a(-\ell) \cdot f_i^b(-\ell) - f_i^c(-\ell))_{\ell \in \{0, \dots, t/2\}}$. P_i also verifies that $f^{E(0)}(-\ell) = 0$ holds for each $\ell \in \{0, \dots, t/2\}$. If not, then P_i broadcasts

complaint(i).

- ii. Parties construct a binary vector $(\text{pub}_1, \dots, \text{pub}_n)$ where $\text{pub}_i = 1$ if P_i broadcasted complaint(i).
- iii. Parties invoke $\mathcal{F}_{\text{pubRec}}$ (Functionality 4.6.5) five times where each P_j inputs $(\text{pub}_1, \dots, \text{pub}_n)$, ZEROS in each instance and the following respectively in five instances: (i) $(f^{E(1)}(j), \dots, f^{E(n)}(j))$, (ii) $(f^{E(0)}(j), \dots, f^{E(0)}(j))$, (iii) $(g_j^a(1), \dots, g_j^a(n))$, (iv) $(g_j^b(1), \dots, g_j^b(n))$ and (v) $(g_j^c(1), \dots, g_j^c(n))$.
For any of the above instances:
 - A. If the output is discard, then proceed to Step 5a.
 - B. If the output is (detect, Bad), then set $\text{ZEROS} = \text{ZEROS} \cup \text{Bad}$. If $|\text{ZEROS}| > t$ then go to Step 5a. Otherwise, go to Step 2.
 - C. Otherwise, when the output is proceed, all parties set $f^{E(i)}(x)$, $f^{E(0)}(x)$ and $f_i^a(x)$, $f_i^b(x)$, $f_i^c(x)$ as the respective output obtained in the corresponding five instances of $\mathcal{F}_{\text{pubRec}}$ for each P_i which broadcasted complaint(i).
- iv. For each P_i which complained, parties verify the checks as in Step 4(b)i using the polynomials returned by the instances of $\mathcal{F}_{\text{pubRec}}$. If it does not hold for some complaining party, then go to Step 5a. Otherwise, go to Step 5c.

5. Output:

- (a) **Discard the dealer:** All parties output \perp .
- (b) **Kill this instance:** All parties output kill.
- (c) **Successful:** Each P_i outputs $(f_i^a(x), g_i^a(y), f_i^b(x), g_i^b(y), f_i^c(x), g_i^c(y))$, where $\{(f_i^a(-\ell), f_i^b(-\ell), f_i^c(-\ell))\}_{\ell \in \{0, \dots, t/2\}}$ defines P_i 's shares of the $t/2 + 1$ multiplication triples.

Lemma 4.6.11. *Let $t < n/3$. Protocol 4.6.10, Π_{PVTs} , perfectly securely computes Functionality 4.6.9, $\mathcal{F}_{\text{PVTs}}$, in the $(\mathcal{F}_{\text{PSS}}, \mathcal{F}_{\text{privRec}}, \mathcal{F}_{\text{pubRec}})$ -hybrid model (Functionality 4.4.10, 4.6.1, 4.6.5), in the presence of a malicious adversary, controlling at most $t < n/3$.*

Proof. We consider the case of an honest dealer and a corrupt dealer separately.

The case of an honest dealer. The simulator is as follows:

1. Invoke \mathcal{A} on an auxiliary input z .
2. Set $\text{SECRETS}_a, \text{SECRETS}_b, \text{SECRETS}_c$ arbitrarily as input (say, all zeros) and run the protocol where the dealer holds $\text{SECRETS}_a, \text{SECRETS}_b, \text{SECRETS}_c$ and all other parties

have no inputs. In particular, simulate the inner functionalities \mathcal{F}_{PSS} , $\mathcal{F}_{\text{privRec}}$ and $\mathcal{F}_{\text{pubRec}}$ as a functionality would run it.

3. Let $A(x, y), B(x, y), C(x, y)$ be the polynomials used in the simulated functionality \mathcal{F}_{PSS} in the last iteration (by iteration, we mean running the protocol from Step 2 until restarting or concluding the protocol). Send $A(x, i), A(i, y), B(x, i), B(i, y)$ and $C(x, i), C(i, y)$ to the functionality for every $i \in I$.
4. If the output of simulated functionality $\mathcal{F}_{\text{privRec}}$ is `kill`, then send `kill` to the functionality.

We will now show that the output of the real and ideal executions are the same. For this, similar to the proof of Π_{PSS} (Lemma 4.4.12) consider the following games:

- **Game₁**: This is the real execution. We run the protocol where the honest dealer uses $\text{SECRETS}_a, \text{SECRETS}_b, \text{SECRETS}_c$ as its input. The output of this game is the view of the adversary and the output of all honest parties in the protocol.
- **Game₂**: Here, we run a modified ideal model, in which the simulator receives the same input = $(\text{SECRETS}_a, \text{SECRETS}_b, \text{SECRETS}_c)$ as in Game₁ as an advice, and the dealer uses input as its input to the functionality. The simulator uses input as its input instead of all zeros. The simulator runs the protocol where the input of the honest dealer is input, exactly as the real execution in Game₁. We claim that the dealer is never discarded. Then, the simulator sends to the functionality the output shares of the corrupted parties in the simulated execution and a message `kill` if some party complains in the simulated execution. If the latter holds, then the functionality sends `kill` to all. Otherwise, the functionality chooses some random polynomials $A(x, y), B(x, y), C(x, y)$ that agree with the output shares of the adversary, and gives the honest parties their shares on these polynomials. The output of this experiment is the view of the adversary as determined by the simulator, and the output of all honest parties.
- **Game₃**: This is the ideal execution. In particular, the simulator receives no advice, and runs as in Game₂, but with input $\text{SECRETS}_a, \text{SECRETS}_b, \text{SECRETS}_c$ set to 0.

We will now show that the output of all the games are identically distributed.

The outputs of Game₁ and Game₂ are identically distributed. The simulator in Game₂ runs the exact same protocol as the real execution in Game₁, and therefore the view of the adversary is identical in both executions. We now turn to the output of the honest parties. We claim that in that execution, the honest dealer is never discarded. In particular:

1. An honest dealer always chooses bivariate polynomials that satisfy the conditions of Functionality 4.4.10 and therefore is not discarded.
2. Similarly, by the guarantees of Functionality 4.6.1, we have that an honest dealer is never discarded. Moreover, $\text{Bad} \subseteq I$ and $\text{Bad} \cup \text{ZEROS} \subseteq I$ and hence $|\text{ZEROS}| \leq t$ always holds and the dealer is not discarded.
3. Finally, for the same reasons as above, during the invocation of Functionality 4.6.5, we have that an honest dealer is never discarded.

Further, if the output of honest parties in Game_1 is `kill`, then it must hold that the output of simulated honest parties in Game_2 is also `kill` and hence the simulator sends `kill` to the functionality. All the honest parties in Game_2 also output `kill`. Finally, when the dealer is honest and parties do not output `kill`, all parties output shares on the same bivariate polynomials, which are $A(x, y), B(x, y), C(x, y)$ that the dealer used in that iteration. In Game_1 , the output of all honest parties is shares on these polynomials. In Game_2 , the simulator sends the shares $(A(x, i), A(i, y))_{i \in I}$, $(B(x, i), B(i, y))_{i \in I}$ and $(C(x, i), C(i, y))_{i \in I}$ to the functionality, the functionality samples new polynomials $A'(x, y), B'(x, y), C'(x, y)$ under the constraints that $A'(x, i) = A(x, i), A'(i, y) = A(i, y), B'(x, i) = B(x, i), B'(i, y) = B(i, y)$ and $C'(x, i) = C(x, i), C'(i, y) = C(i, y)$ for every $i \in I$, and $\text{SECRETS}_a, \text{SECRETS}_b, \text{SECRETS}_c$ are embedded in $A'(x, y), B'(x, y), C'(x, y)$ respectively. The output of all honest parties is then equivalent to just outputting shares on $A'(x, y), B'(x, y), C'(x, y)$. Using a similar argument as Claim 4.4.13, it can be seen that the output of honest parties is identically distributed.

The outputs of Game_2 and Game_3 are identically distributed. The only difference between the two games is that in Game_2 the simulator uses the same secrets input as the honest dealer uses in the ideal execution, whereas in Game_3 the honest dealer uses input as all 0. The following claim shows that the shares that the corrupted parties receives in the simulated execution is identically distributed. In both execution, given the shares and the message `kill` that the simulator sends to the functionality, the outputs of the honest parties are defined in exactly the same process (that is, the functionality either sends `kill` to all or uses $\text{SECRETS}_a, \text{SECRETS}_b, \text{SECRETS}_c$ and the shares sent by the adversary to define the shares of honest parties). Therefore it is enough to show that the view of the adversary is identically distributed. This can be easily shown using similar argument as Claim 4.4.14.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} on the auxiliary input z .
2. Observe that all honest parties have no input to the protocol. Simulate running the

protocol with the adversary, while also simulating Functionalities 4.4.10, 4.6.1 to the adversary as the functionality would run it.

3. If the output of some simulated honest party is \perp , then send $A(x, y) = B(x, y) = C(x, y) = x^{3t/2+1}$ to the functionality, in which case it sends \perp to all parties.
4. If the output of some simulated honest party is `kill`, then send $M = \text{kill}$ to the functionality.
5. Otherwise, let J be a set of $t+1$ honest parties. Reconstruct the unique bivariate polynomials $A(x, y), B(x, y), C(x, y)$ that satisfy $A(x, j) = f_j^a(x), B(x, j) = f_j^b(x)$ and $C(x, j) = f_j^c(x)$ for every $j \in J$, where $f_j^a(x), g_j^a(y), f_j^b(x), g_j^b(y)$ and $f_j^c(x), g_j^c(y)$ is the output of the simulated honest party in the simulated execution. Send $A(x, y), B(x, y), C(x, y)$ to the functionality and halt.

Since the code of each party in the protocol which is not the dealer is deterministic, and the functionalities 4.4.10, 4.6.1 are deterministic, the view of the adversary is *identical* in the real and ideal executions. Moreover, since the functionality is deterministic, we can separately consider the view and the outputs of the honest parties. Thus, all that is left to be shown is that the output of the honest parties is the same in the real and in the ideal executions. We have the following cases to consider:

1. **There exists an honest party that outputs \perp in the real world.** An honest party outputs \perp if and only if one of the following conditions holds: (i) \mathcal{F}_{PSS} (Functionality 4.4.10), $\mathcal{F}_{\text{privRec}}$ or $\mathcal{F}_{\text{pubRec}}$ returns \perp , or (ii) $|\text{ZEROS} \cup \text{Bad}| > t$. In the first case, if $\mathcal{F}_{\text{PSS}}, \mathcal{F}_{\text{privRec}}$ or $\mathcal{F}_{\text{pubRec}}$ returned \perp to one honest party, it implies that all the honest parties received \perp , and thus all would output \perp in the protocol. For the latter case, due to the guarantees of the functionalities $\mathcal{F}_{\text{PSS}}, \mathcal{F}_{\text{privRec}}$ and $\mathcal{F}_{\text{pubRec}}$, all the honest parties must hold identical sets `Bad` and `ZEROS`. Hence, all would output \perp in the protocol. Since the real and simulated executions are identical, the simulated honest parties must also output \perp . In this case, the simulator invokes the functionality with $A(x, y) = B(x, y) = C(x, y) = x^{3t/2+1}$, causing all the honest parties to output \perp in the ideal execution.
2. **There exists an honest party that outputs `kill` in the real world.** An honest party outputs `kill` if and only if $\mathcal{F}_{\text{privRec}}$ returns `kill`. By the guarantees of $\mathcal{F}_{\text{privRec}}$, we have that all the honest parties receive `kill` from $\mathcal{F}_{\text{privRec}}$ and hence all would output `kill` in the protocol. Due to identical views of the honest parties in the real and simulated execution, the same must hold true for each simulated honest party. In this case, the simulator sends `kill` to the functionality causing all the honest parties to receive output

kill in the ideal execution. This is exactly the output of the honest parties in the real world.

3. **No honest party outputs \perp or kill in the real world.** In this case, first note that by the guarantees of \mathcal{F}_{PSS} , we have that all the honest parties P_i output shares $(f_i^a(x), g_i^a(y), f_i^b(x), g_i^b(y), f_i^c(x), g_i^c(y))$ consistent with some bivariate polynomials $A(x, y), B(x, y)$ and $C(x, y)$ respectively. Thus, it remains to show that $A(-\ell, 0) \cdot B(-\ell, 0) = C(-\ell, 0)$ indeed holds for every $\ell \in \{0, \dots, t/2\}$. For this, first note that since no honest party outputs \perp or kill, in the final iteration of the protocol, it must hold that each P_i successfully reconstructs its $f^{E(0)}(x)$ and $f^{E(i)}(x)$ polynomials in Step 4a which are required to verify the product relation. We now claim that since the honest parties do not output \perp in the final iteration of the protocol, it must hold that no honest party P_i broadcast $\text{complaint}(i)$ in Step 4(b)i of the final iteration. Suppose some honest party P_i had indeed broadcast $\text{complaint}(i)$ at this step, all the parties must have invoked $\mathcal{F}_{\text{pubRec}}$ in Step 4(b)iii. Since this is the final iteration, we have that $\mathcal{F}_{\text{pubRec}}$ did not output $(\text{detect}, \text{Bad})$ such that $|\text{Bad}| > t/2$, as otherwise the parties would have rebooted to start a new iteration. Moreover, no party output \perp hence $\mathcal{F}_{\text{pubRec}}$ did not output discard . Consequently, each party would successfully reconstructed the f polynomials of the complaining party P_i . Given that the reconstructed polynomials agree with the shares of at least $n - t/2 \geq 5t/2 + 1$ parties, it must agree with with the shares of at least $3t/2 + 1$ honest parties (since at most $t/2$ parties may have been identified as Bad in $\mathcal{F}_{\text{pubRec}}$), thus forcing the reconstruction of the dealer's polynomial committed during the sharing phase. Thus, all the parties would publicly verify an honest P_i 's complain and as a result discard the dealer, which is a contradiction. Hence, it must hold that no honest party P_i broadcast $\text{complaint}(i)$ in Step 4(b)i of the final iteration. Since no honest party complains, it must hold that $A(-\ell, i) \cdot B(-\ell, i) - C(-\ell, i) = E_{-\ell}(i)$ holds for each honest P_i and every $\ell \in \{0, \dots, t/2\}$. Since each of $A(-\ell, y) \cdot B(-\ell, y) - C(-\ell, y)$ and $E_{-\ell}(y)$ are at most degree $2t$ polynomials, which agree in at least $2t + 1$ points corresponding to the honest parties, it must hold that $A(-\ell, y) \cdot B(-\ell, y) - C(-\ell, y) = E_{-\ell}(y)$. Using a similar argument, it can be clearly seen that $E_{-\ell}(0)$ holds when the dealer is not discarded. Thus, it can be concluded that $A(-\ell, 0) \cdot B(-\ell, 0) = C(-\ell, 0)$ holds. In the simulated execution, the simulator chooses an arbitrary set of $t + 1$ honest parties and reconstructs the bivariate polynomials that agree with their output. Since the real and simulated executions are identical, it must hold that these polynomials are $A(x, y), B(x, y)$ and $C(x, y)$ respectively. The simulator then invokes the functionality with these polynomials causing the honest parties in the ideal execution to output shares

that are identical to those in the real execution. □

Lemma 4.6.12. *Let $t < n/3$ and $d \leq t/4$. There exists a protocol that implements Functionality 4.6.9, $\mathcal{F}_{\text{PVTs}}$, has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast for sharing $\mathcal{O}(n)$ multiplication triples simultaneously in $\mathcal{O}(1)$ rounds. Every party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

4.7 Batched Verifiable Triple Sharing

In this section, we discuss how to run m instances of our packed VTS while keeping the broadcast cost unchanged. Specifically, while one instance of the packed VTS requires $\mathcal{O}(n^2 \log n)$ bits communication over point-to-point channels and each party broadcasts $\mathcal{O}(n \log n)$ bits, we show how to run m instances together at a cost of $\mathcal{O}(mn^2 \log n)$ bits over point-to-point channels and retain the same broadcast of $\mathcal{O}(n \log n)$ bits per party. An important property to note here is that although an adversary can compromise an instance of packed VTS even for an honest dealer, in the batched version we ensure that at most n out of the m instances can be compromised. When $m > n$, our protocols achieves the properties of a verifiable triple sharing. That is, when the dealer is honest, our protocol guarantees privacy of triples obtained from the $m - n$ uncompromised instances of packed VTS. While for a corrupt dealer, as in packed VTS, our zero knowledge proof ensures the correctness of the triples. We now elaborate on the changes necessary to the packed VTS protocol and its sub-protocols.

Dealing the shares of triples and product polynomials. The dealer holds m lists of triples, and one set $\text{ZEROS} \subset [n]$. Further, for the parties in ZEROS , the shares of on all the bivariate polynomials are assumed to be 0. For sharing the triples as well as the coefficients of the product polynomials of m instances, batched variant of the packed secret sharing $\mathcal{F}_{\text{PSS}}^{\text{batched}}$ (Functionality 4.5.4) is invoked. This ensures a cost of $\mathcal{O}(mn^2 \log n)$ bits over point-to-point channels and a broadcast of $\mathcal{O}(n \log n)$ bits per party.

Batched private reconstruction. The change in the protocol here follows closely to the batched reconstruction of g polynomials defined in Section 4.5. Specifically, in Step 2, a party P_ℓ may fail to reconstruct h_0 or h_ℓ in multiple instances. However, it suffices for P_ℓ to choose one instance (say the instance with a minimum index β) and complain with respect to β . The complaint for a party now looks like $\text{complaint}(\ell, \beta)$. Following this, the public verification happens only for the β th instance for P_ℓ . In the case that the dealer is not discarded and the publicly identified set Bad has at most $t/2$ parties, each party is guaranteed to reconstruct its

polynomials in all the instances successfully. This follows similar to the argument described in Section 4.5. Specifically, using the publicly identified Bad set, and in addition the locally identified conflicts (by comparing the values it received over point-to-point channels and those broadcast by parties during public reveal), a party can recognize more than $t/2$ errors and correct the remaining (less than $t/2$) errors across the m instances. We provide the complete modelling via $\mathcal{F}_{\text{privRec}}^{\text{batched}}$ (Functionality 4.7.1). The protocol $\Pi_{\text{privRec}}^{\text{batched}}$ (Protocol 4.7.2) and its security proof follow.

Functionality 4.7.1: Batched Private Reconstruction of $3t/2$ -Shared Polynomials – $\mathcal{F}_{\text{privRec}}^{\text{batched}}$

- **Input:** All honest parties send $\text{ZEROS} \subset [n]$. When the dealer is honest, $\text{ZEROS} \subseteq I$. Each honest party inputs shares on m sets of polynomials, each of degree $3t/2$ to $\mathcal{F}_{\text{privRec}}^{\text{batched}}$. It reconstructs the polynomials as $\{h_0^k(x), \dots, h_n^k(x)\}_{k \in [m]}$.
 - **The functionality:**
 1. If the dealer is honest, then send (a) the polynomials $h_0^k(x), \dots, h_n^k(x)$ for every $k \in [m]$ to the dealer; (b) ZEROS , $(h_0^k(i), \dots, h_n^k(i))_{i \in I}$ and $(h_0^k(x), h_i^k(x))_{i \in I}$ for each $k \in [m]$ to the adversary. If the dealer is corrupted, then send ZEROS and $h_0^k(x), \dots, h_n^k(x)$ to the adversary for each $k \in [m]$.
 2. Receive a binary vector leak of length m with at most n 1's from the adversary. For every k such that $\text{leak}_k = 1$, send $\{h_0^k(x), \dots, h_n^k(x)\}$ to the adversary.
 3. Receive a message M from the adversary.
 4. **Output:**
 - (a) If $M = \text{discard}$ and the dealer is corrupted, then send discard to all parties.
 - (b) If $M = (\text{detect}, \text{Bad})$ with $\text{Bad} \cap \text{ZEROS} = \emptyset$ and $|\text{Bad}| > t/2$, and in case of an honest dealer then $\text{Bad} \subseteq I$, then send $(\text{detect}, \text{Bad})$ to all parties.
 - (c) If $M = \text{kill}$ and leak is not a 0-vector then then send $(\text{proceed}, h_0^k(x), h_\ell^k(x))$ to each party P_ℓ for every k where such that $\text{leak}_k = 0$. For every other k , send kill to all the parties.
 - (d) If $M = \text{proceed}$ and leak is a 0-vector then send $(\text{proceed}, h_0^k(x), h_\ell^k(x))$ for every $k \in [m]$ to each party P_ℓ .
-

Protocol 4.7.2: Batched Private Reconstruction of $3t/2$ -Shared Polynomials – $\Pi_{\text{privRec}}^{\text{batched}}$

- **Input:** All parties hold the same set ZEROS . Each honest party P_j holds m sets of shares $(h_0^k(j), \dots, h_n^k(j))$ for each $k \in [m]$ on polynomials $h_0^k(x), \dots, h_n^k(x)$ each of degree

$3t/2$. It is guaranteed that all the shares of honest parties lie on the same $3t/2$ degree polynomials.

• **The protocol:**

1. Each P_j sends $(j, h_0^k(j), h_\ell^k(j))$ for every $k \in [m]$ to every P_ℓ .
2. Let (j, u_j^k, v_j^k) be the value P_ℓ received from P_j . For every $j \in \text{ZEROS}$, (j, u_j^k, v_j^k) is such that $u_j = v_j = 0$. Given all (j, u_j^k) and (j, v_j^k) , P_ℓ looks for a codeword of distance at most $t/2$ from all the values it received (see Corollary 4.3.3, item 1). If there is such a codeword, set $h_0^k(j)$ and $h_\ell^k(x)$ to be the unique Reed-Solomon reconstruction respectively for each $k \in [m]$. If there is no such a unique codeword for some k , then P_ℓ broadcasts $\text{complaint}(\ell, \beta)$ where β is the minimal index from $\{1, \dots, m\}$ and every party P_j broadcasts $\text{reveal}(\ell, \beta, j, h_0^\beta(j), h_\ell^\beta(j))$. If P_ℓ sees some $\text{reveal}(\ell, \beta, j, u_j^\beta, v_j^\beta)$ with a value different than the one sent by P_j then add j to localBad_ℓ .
3. If no party broadcasts $\text{complaint}(\ell, \beta)$ then go to Step 9d.
4. The dealer sets $\text{Bad} = \emptyset$. For each $\text{reveal}(\ell, \beta, j, u, v)$ message broadcasted, the dealer verifies that $u = h_0^\beta(j)$ and $v = h_\ell^\beta(j)$. If not, then it adds j to Bad . The dealer broadcasts Bad .
5. Verify that (i) $|\text{ZEROS} \cup \text{Bad}| \leq t$; and (ii) $\text{Bad} \subset [n] \setminus \text{ZEROS}$. Otherwise, discard – go to Step 9a.
6. If $|\text{Bad}| > t/2$ then there is a large detection – go to Step 9b.
7. Otherwise, consider all the points $R_\ell = \{(j, u_j^\beta)\}$ and $T_\ell = \{(j, v_j^\beta)\}$ such that $\text{reveal}(\ell, \beta, j, u_j^\beta, v_j^\beta)$ was broadcasted in Step 2 for $j \notin \text{Bad}$, where $u_j^\beta = v_j^\beta = 0$ if $j \in \text{ZEROS}$. Verify that R_ℓ and T_ℓ each define a unique polynomial of degree $3t/2$. If not, go to Step 9a.
8. If the dealer is not publicly discarded, then each P_ℓ sets $h_0^k(x)$ and $h_\ell^k(x)$ for each $k \in [m]$ as the unique decoding of the points (j, u_j^k) and (j, v_j^k) respectively (see Corollary 4.3.3, item 2) for every $j \notin \text{Bad} \cup \text{localBad}_\ell$, received in Step 1 and proceed to Step 9c.
9. **Output:**
 - (a) Discard the dealer: Output discard .
 - (b) Detect: Output $(\text{detect}, \text{Bad})$.
 - (c) Leak: Output $(\text{proceed}, h_0^k(x), h_\ell^k(x))$ for each $k \in [m]$ such that $\text{complaint}(\cdot, k)$ was not broadcasted in Step 2. For every other k , output kill .
 - (d) Proceed: Each P_ℓ outputs $(\text{proceed}, h_0^k(x), h_\ell^k(x))$ for every $k \in [m]$.

Lemma 4.7.3. Protocol 4.7.2, $\Pi_{\text{privRec}}^{\text{batched}}$ perfectly securely computes Functionality 4.7.1, $\mathcal{F}_{\text{privRec}}^{\text{batched}}$, in the presence of a malicious adversary controlling at most $t < n/3$.

Proof. We show the case of an honest dealer and a corrupted dealer separately.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z .
2. Receive from the functionality the sets ZEROS, $\{(h_0^k(i), \dots, h_n^k(i))\}_{i \in I}$ and $(h_0^k(x), h_i^k(x))_{i \in I}$ for every $k \in [m]$.
3. For each $i \in I$, simulate every honest P_j sending $(j, h_0^k(j), h_i^k(j))$ to P_i . Receive $(i, h_0^k(i), h_j^k(i))$ from the adversary for each $i \in I$ and every honest P_j , for every $k \in [m]$.
4. If for some honest P_j , (i, u_i^k, v_i^k) sent by the adversary is such that $u_i^k \neq h_0^k(i)$ or $v_i^k \neq h_j^k(i)$ for more than $t/2$ such $i \in I$ then simulate P_j broadcasting $\text{complaint}(j, \beta)$ where β is the minimal index from $[m]$. Also listen to all $\text{complaint}(i, \beta)$ messages broadcasted by the adversary. Construct a vector leak of length m such that $\text{leak}_\beta = 1$ for each β for which $\text{complaint}(\cdot, \beta)$ was broadcasted by some party, and send leak to the functionality. Receive $\{h_0^\beta(x), \dots, h_n^\beta(x)\}$ from the functionality for each β where $\text{leak}_\beta = 1$. Set $\text{Bad} = \emptyset$.
5. For each $\text{complaint}(\ell, \beta)$ broadcasted by some P_ℓ , simulate broadcasting $(\ell, \beta, j, h_0^\beta(j), h_\ell^\beta(j))$ for every honest P_j and listen to all the adversary's broadcasts.
6. For each $i \in I$ where $(\ell, \beta, i, u_i^\beta, v_i^\beta)$ broadcasted is such that $u_i^\beta \neq h_0^\beta(i)$ or $v_i^\beta \neq h_\ell^\beta(i)$, add i to Bad .
7. Simulate the dealer broadcasting Bad . If $|\text{Bad}| > t/2$ then send $(\text{detect}, \text{Bad})$ to the functionality and halt.
8. If leak is not a 0-vector then send kill to the functionality and halt.
9. Otherwise, send proceed to the functionality and halt.

The protocol as well as the simulator is deterministic. Hence, it can be easily observed that the view of the adversary in the real and ideal executions is identical. It remains to show that the output of honest parties is identical in the real and ideal world.

Towards that end, note that in the real world, an honest dealer is discarded if and only if one of the following conditions holds:

1. $|\text{ZEROS} \cup \text{Bad}| > t$.

2. $\text{Bad} \not\subseteq [n] \setminus \text{ZEROS}$.
3. R_ℓ or T_ℓ do not define a unique polynomial of degree- $3t/2$.

Note that for an honest dealer, an honest party never belongs to Bad and we have that $\text{ZEROS} \subseteq I$. It is clear that none of the above conditions hold and the dealer is not discarded. We thus have the following cases to consider:

1. **There exists an honest party that outputs $(\text{detect}, \text{Bad})$:** In this case, it must hold that $|\text{Bad}| > t/2$. Since the corresponding message was broadcasted by the dealer, it must hold that all the honest parties output $(\text{detect}, \text{Bad})$. The simulator emulates the interaction of the honest parties with the dealer as in the real execution, hence all the simulated honest parties hold the same set Bad . In that case, the simulator sends $(\text{detect}, \text{Bad})$ to the functionality, causing all the honest parties in the ideal world to output the same.
2. **There exists an honest party that outputs kill for some $k \in [m]$:** In this case, it must hold that $|\text{Bad}| \leq t/2$. Moreover, there exists some party P_ℓ which broadcasted $\text{complaint}(\ell, k)$ and each R_ℓ and T_ℓ define a unique degree $3t/2$ polynomials. Since all the corresponding messages are broadcast, all the honest parties would output kill for k . The same holds true for each k for which $\text{complaint}(\cdot, k)$ was broadcast. For every other k , it must hold that each honest party P_ℓ is able to reconstruct $h_0^k(x), h_\ell^k(x)$ either in Step 2 or in Step 8 after discarding the shares of (at least $t/2$ corrupt) parties in $\text{Bad} \cup \text{localBad}_\ell$ (similar to batched reconstruction of g polynomials in Section 4.5). Since these polynomials are degree $3t/2$ and agree with the shares of at least $2t + 1$ honest parties, they are guaranteed to be the same polynomials defined by the input shares of the honest parties. The simulator emulates the interaction of the honest parties with the dealer as in the real execution, hence all the simulated honest parties P_ℓ hold the same set Bad , see the same complaints. In this case, the simulator constructs leak with $\text{leak}_k = 1$ for each k such that $\text{complaint}(\cdot, k)$ was broadcast and sends leak to the functionality, followed by kill , causing all the honest parties in the ideal execution to have the same output as the honest parties in the real execution.
3. **There exists an honest party that outputs proceed for all $k \in [m]$:** This implies that no party broadcast complaint in the real execution. Thus, it must hold that each honest party P_ℓ obtained a unique reconstruction in Step 2, which agrees with the shares of all the honest parties. Since the reconstructed polynomials and $h_0^k(x), h_\ell^k(x)$ defined by the honest parties' input shares are each of degree $3t/2$ and agree in at least $2t + 1$ points, it must hold that the unique reconstructed polynomials are $h_0^k(x), h_\ell^k(x)$. Hence

it must hold that all honest parties output $(\text{proceed}, h_0^k(x), h_\ell^k(x))$ for every $k \in [m]$ in the real execution. Since the simulated execution is identical to the real execution, all the simulated honest parties also see that no complaint was broadcast. In this case, the simulator sends proceed to the functionality, causing all the honest parties in the ideal world to have an output that is identical to the output of the honest parties in the real world.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary \mathcal{A} with an auxiliary input z .
2. Receive from the functionality the set ZEROS, and the polynomials $h_0^k(x), \dots, h_n^k(x)$ for each $k \in [m]$.
3. Simulate the protocol where each honest party P_j holds $h_0^k(j), \dots, h_n^k(j)$ for each k and all parties have the same set ZEROS.
4. If some party P_ℓ broadcasts $\text{complaint}(\ell, \beta)$ in Step 2 then the simulator sets $\text{leak}_\beta = 1$. It sends $\text{leak} = (\text{leak}_1, \dots, \text{leak}_m)$ to the functionality.
5. Send the message M to the functionality according to the following cases (the proof will show that the cases are mutually-exclusive):
 - (a) If the output of some simulated honest party is discard , then send discard to the functionality and halt.
 - (b) If the output of some simulated honest party is $(\text{detect}, \text{Bad})$, then send $(\text{detect}, \text{Bad})$ to the functionality and halt.
 - (c) If the output of some simulated honest party is kill for some k then send kill to the functionality and halt.
 - (d) If the output of some simulated honest party is proceed for all k , then send proceed to the functionality and halt.

Since the simulator uses the exact same inputs of the simulated honest parties as the real honest parties in the real execution, and since the protocol is deterministic, we get that the view of the adversary is exactly the same in the real and in the ideal executions. Thus it remains to be shown that the output of the honest parties is identical in the real and ideal executions. We have the following cases to consider:

1. **There exists an honest party that outputs discard in the real world:** Observe that an honest party outputs discard if and only if one of the following conditions holds.

- (a) The dealer broadcasted Bad such that either (i) $|\text{ZEROS} \cup \text{Bad}| > t$; or (ii) $\text{Bad} \not\subset [n] \setminus \text{ZEROS}$. Since all honest parties hold the same set ZEROS , and the set Bad is broadcasted, we get that all honest parties output `discard`.
- (b) The dealer broadcasted Bad with $|\text{Bad}| \leq t/2$, and for which $|\text{ZEROS} \cup \text{Bad}| \leq t$ and $\text{Bad} \subset [n] \setminus \text{ZEROS}$. However, for $R_\ell = \{(j, u_j^\beta)\}$ and $T_\ell = \{(j, v_j^\beta)\}$ such that $\text{reveal}(\ell, \beta, j, u_j^\beta, v_j^\beta)$ was broadcasted in Step 2 and $j \notin \text{Bad}$, it holds that R_ℓ or T_ℓ does not define a unique polynomial of degree $3t/2$. Since the set R_ℓ and T_ℓ are public, all honest parties will identify that there is no unique reconstruction, and all would output `discard`.

Since the simulated honest parties have the same view as the honest parties, the simulated honest parties also output `discard`. In this case, the simulator sends `discard` to the functionality causing all the honest parties to output `discard` in the ideal execution.

2. **There exists an honest party that outputs `(detect, Bad)` in the real world:** In this case, it must hold that $|\text{ZEROS} \cup \text{Bad}| \leq t$ and $\text{Bad} \subset [n] \setminus \text{ZEROS}$. Moreover, it must hold that $|\text{Bad}| \geq t/2$. Since the corresponding set Bad is broadcast, all the honest parties hold the same set and hence output `(detect, Bad)`. The simulated honest parties hold an identical output, and thus the simulator sends `(detect, Bad)` to the functionality, which in turn sends the same to all the honest parties in the ideal execution.
3. **There exists an honest party that outputs `kill` for some $k \in [m]$ in the real world:** In this case, it must hold that some party P_ℓ broadcast `complaint`(ℓ, k). Moreover, $|\text{Bad}| \leq t/2$ and R_ℓ and T_ℓ each define a unique degree $3t/2$ polynomials. Since all the corresponding messages are broadcast, it must hold that all the honest parties output `kill` for k . This must hold for each k for which `complaint`(\cdot, k) was broadcast by some party. Further, for every other k , it must hold that each honest party P_ℓ is able to reconstruct $h_0^k(x), h_\ell^k(x)$ either in Step 2 or in Step 8 after discarding the shares of (at least $t/2$) parties in $\text{Bad} \cup \text{localBad}_\ell$ (similar to batched reconstruction of g polynomials in Section 4.5). Note that an honest party never belongs to localBad_ℓ set of another honest party P_ℓ . Hence, during the reconstruction of its polynomials, a party may discard shares of at most $t/2$ honest parties from Bad . Since these reconstructed polynomials are degree $3t/2$ and agree with the shares of at least $2t + 1 - t/2 \geq 3t/2 + 1$ honest parties, they are guaranteed to be the same polynomials defined by the input shares of the honest parties. The simulator emulates the honest parties as in the real execution, hence all the simulated honest parties P_ℓ see the same complaints. In this case, the simulator constructs `leak` with $\text{leak}_k = 1$ for each k such that `complaint`(\cdot, k) was broadcast

and sends `leak` to the functionality, followed by `kill`, causing all the honest parties in the ideal execution to have the same output as the honest parties in the real execution.

4. **There exists an honest party that outputs `proceed` for all $k \in [m]$ in the real world:** In this case, we show that all honest parties P_ℓ output (`proceed`, $h_0^k(x)$, $h_\ell^k(x)$) where $h_0^k(x)$, $h_\ell^k(x)$ are the degree $3t/2$ polynomials that are interpolated from the respective input shares of the honest parties and are guaranteed to exist under our input assumption. Towards that, observe that since there exists an honest party that does not output `discard`, (`detect`, `Bad`) or `kill` for any k it must hold that no party broadcasted complaint in Step 2. Hence, it must hold that each honest P_ℓ has a unique reconstruction and consequently lesser than $t/2$ errors occurred in the reconstruction for P_ℓ . The reconstructed polynomials agree with the shares of at least $n - t/2 \geq 5t/2 + 1$ parties, hence they agree with the shares of at least $3t/2 + 1$ honest parties. Since the reconstructed polynomials and each $h_0^k(x)$, $h_\ell^k(x)$ are of degree $3t/2$, it must hold that the reconstructed polynomials for each honest P_ℓ are $h_0^k(x)$, $h_\ell^k(x)$ consistent with the input shares of the honest parties. In this case, the simulated honest parties also observe that no party complains and hence the simulator sends `proceed` to the functionality, causing all the honest parties in the ideal world to obtain an output identical to the output in the real execution.

□

Lemma 4.7.4. *Let $t < n/3$. There exists a protocol that implements Functionality 4.7.1 and has a communication complexity of $\mathcal{O}(mn^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast in $\mathcal{O}(1)$ rounds. Every party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

Note that in the batched private reconstruction, the public verification happens for one instance corresponding to each party's complaint. Thus, in total, we have that the public reveal of shares is performed for at most n instances out of the total m that are batched together. Consequently, for an honest dealer, we have that the adversary does not learn any additional information for the $m - n$ instances where the shares are not publicly revealed. Since private reconstruction is used for reconstructing distinct points on the product polynomials to each party in the triple sharing protocol, we have the same privacy guarantee of $m - n$ uncompromised instances carry over to the batched packed VTS protocol.

Public reconstruction for complaining parties. Similar to private reconstruction, if the product relation fails to hold during the verification in Step 4b, a party P_i complains only for one instance, say β_i . The public verification proceeds as before via $\mathcal{F}_{\text{pubRec}}$ (Functionality 4.6.5), with the only difference being that the i th input share of every party corresponds

to the shares of polynomials in β_i th instance, for which P_i broadcasts a complaint. At the conclusion of this verification, if the dealer is not discarded then parties either detect more than $t/2$ conflicts with the dealer, or successfully reconstruct the polynomials of P_i for the β_i th instance. In the latter case, the verification of product polynomial is performed on the reconstructed polynomial and the dealer is discarded in case of failure. Note that when the dealer is corrupt, it is sufficient for an honest party to complain in one instance. The “binding” property of $\mathcal{F}_{\text{PSS}}^{\text{batched}}$ guarantees that for any individual instance, if a polynomial is reconstructed then it is the same polynomial committed to by the dealer in the sharing phase. Thus if the dealer shares incorrect multiplication triples, causing an honest party to broadcast complaint in some β_i th instance such that its polynomials are reconstructed successfully during Step 4b, then the dealer is guaranteed to be discarded. The functionality for batched verifiable secret sharing appears below. We provide the protocol for completeness.

Functionality 4.7.5: Batched and Packed Verifiable Triple Sharing Functionality – $\mathcal{F}_{\text{PVTS}}^{\text{batched}}$

The functionality is parameterized by a set of corrupted parties $I \subset [n]$.

1. Honest dealer:

- (a) The dealer sends $\{\text{SECRETS}_a^\alpha, \text{SECRETS}_b^\alpha, \text{SECRETS}_c^\alpha\}_{\alpha \in [m]}$ to $\mathcal{F}_{\text{PVTS}}^{\text{batched}}$.
 - (b) The adversary sends $(f_i^{\alpha,a}(x), g_i^{\alpha,a}(y))_{i \in I}, (f_i^{\alpha,b}(x), g_i^{\alpha,b}(y))_{i \in I}, (f_i^{\alpha,c}(x), g_i^{\alpha,c}(y))_{i \in I}$ to $\mathcal{F}_{\text{PVTS}}^{\text{batched}}$ such that $f_i^{\alpha,a}(k) = g_k^{\alpha,a}(i)$, $f_i^{\alpha,b}(k) = g_k^{\alpha,b}(i)$ and $f_i^{\alpha,c}(k) = g_k^{\alpha,c}(i)$ for every $i, k \in I$ and every $\alpha \in [m]$.
 - (c) The functionality chooses random bivariate polynomials $A^\alpha(x, y)$, $B^\alpha(x, y)$ and $C^\alpha(x, y)$ of degree $3t/2$ in x and t in y under the constraints that (i) $\text{SECRETS}_a^\alpha, \text{SECRETS}_b^\alpha, \text{SECRETS}_c^\alpha$ are embedded in $A^\alpha, B^\alpha, C^\alpha$ respectively; (ii) $A^\alpha(x, i) = f_i^{\alpha,a}(x)$, $B^\alpha(x, i) = f_i^{\alpha,b}(x)$ and $C^\alpha(x, i) = f_i^{\alpha,c}(x)$ for every $i \in I$; (iii) $A^\alpha(i, y) = g_i^{\alpha,a}(y)$, $B^\alpha(i, y) = g_i^{\alpha,b}(y)$ and $C^\alpha(i, y) = g_i^{\alpha,c}(y)$ for every $i \in I$.
- 2. Corrupted dealer:** The dealer sends $A^\alpha(x, y)$, $B^\alpha(x, y)$ and $C^\alpha(x, y)$ for each $\alpha \in [m]$ to $\mathcal{F}_{\text{PVTS}}^{\text{batched}}$ that verifies that (i) $A^\alpha(x, y)$, $B^\alpha(x, y)$ and $C^\alpha(x, y)$ are of degree $3t/2$ in x and degree t in y for each $\alpha \in [m]$; and (ii) $A(-i, 0) \cdot B(-i, 0) = C(-i, 0)$ holds for each $i \in \{0, \dots, t/2\}$. If not, $\mathcal{F}_{\text{PVTS}}^{\text{batched}}$ replaces each $A^\alpha(x, y)$, $B^\alpha(x, y)$ and $C^\alpha(x, y)$ with \perp .
- 3. Output:** $\mathcal{F}_{\text{PVTS}}^{\text{batched}}$ sends to each party P_j the pairs of polynomials $A^\alpha(x, j), A^\alpha(j, y)$, $B^\alpha(x, j), B^\alpha(j, y)$ and $C^\alpha(x, j), C^\alpha(j, y)$ for every $\alpha \in [m - n]$.
-

Protocol 4.7.6: Batched and Packed VTS in the $(\mathcal{F}_{\text{PSS}}, \mathcal{F}_{\text{privRec}}^{\text{batched}}, \mathcal{F}_{\text{pubRec}})$ -hybrid model –

$\Pi_{\text{PVTS}}^{\text{batched}}$

Input: The dealer holds m sets of three lists $\text{SECRETS}_a^\alpha = \{a_0^\alpha, \dots, a_{t/2}^\alpha\}$, $\text{SECRETS}_b^\alpha = \{b_0^\alpha, \dots, b_{t/2}^\alpha\}$, $\text{SECRETS}_c^\alpha = \{c_0^\alpha, \dots, c_{t/2}^\alpha\}$, each of size $t/2 + 1$ such that $c_i^\alpha = a_i^\alpha b_i^\alpha$ holds for each $i \in \{0, \dots, t/2\}$ and each $\alpha \in [m]$.

The protocol:

1. All parties set $\text{ZEROS} = \emptyset$.
2. **Dealing the shares of triples:** Parties invoke $\mathcal{F}_{\text{PSS}}^{\text{batched}}$ (Functionality 4.5.4) three times with $d = 0$. The dealer inputs $(\text{SECRETS}_a^\alpha)_{\alpha \in [m]}$, $(\text{SECRETS}_b^\alpha)_{\alpha \in [m]}$, $(\text{SECRETS}_c^\alpha)_{\alpha \in [m]}$ respectively and each party inputs ZEROS . If the output of any instance is \perp , then proceed to Step 5a. Otherwise, each P_i holds $f_i^{\alpha,a}(x) = A^\alpha(x, i)$, $g_i^{\alpha,a}(y) = A^\alpha(i, y)$, $f_i^{\alpha,b}(x) = B^\alpha(x, i)$, $g_i^{\alpha,b}(y) = B^\alpha(i, y)$ and $f_i^{\alpha,c}(x) = C^\alpha(x, i)$, $g_i^{\alpha,c}(y) = C^\alpha(i, y)$.
3. **Dealing the shares of product polynomials:**
 - (a) For each $\ell \in \{0, \dots, t/2\}$ and every $\alpha \in [m]$, the dealer defines the polynomials $E_{-\ell}^\alpha(y)$ of degree at most $2t$ such that $E_{-\ell}^\alpha(y) = A^\alpha(-\ell, y) \cdot B^\alpha(-\ell, y) - C^\alpha(-\ell, y) = e_{(-\ell,0)}^\alpha + e_{(-\ell,1)}^\alpha y + \dots + e_{(-\ell,2t)}^\alpha y^{2t}$. Define $e_i^\alpha = (e_{(-t/2,i)}^\alpha, \dots, e_{(0,i)}^\alpha)$ for $i \in \{0, \dots, 2t\}$, as the vector of i th coefficients of all the $t/2 + 1$ polynomials for the α th set of polynomials.
 - (b) The dealer views the coefficients of these $t/2 + 1$ polynomials for each of the m sets of polynomials as (at most) eight matrices $\text{SECRETS}_1^\alpha, \dots, \text{SECRETS}_8^\alpha$, each of size $(t/2 + 1)(t/4 + 1)$. Specifically, $\text{SECRETS}_u^\alpha(\cdot, b) = e_{(t/4+1) \cdot (u-1) + b}^\alpha$ where $b \in \{0, \dots, t/4\}$ and $u \in [8]$.
 - (c) All the parties invoke $\mathcal{F}_{\text{PSS}}^{\text{batched}}$ (Functionality 4.5.4) (at most) eight times with $d = t/4$, where the dealer inputs SECRETS_u^α for each $u \in [8]$ and every $\alpha \in [m]$ respectively and each party inputs ZEROS . If the output of any instance is \perp , then proceed to Step 5a. Otherwise, each P_i holds the degree- $(t + t/2)$ polynomials $f_i^{\alpha,u}(x) = E_u^\alpha(x, i)$ and degree- $(t + t/4)$ polynomials $g_i^{\alpha,u}(y) = E_u^\alpha(i, y)$ for all $u \in [8]$ and every $\alpha \in [m]$.
 - (d) Let $f^{e_{(t/4+1) \cdot (u-1) + b}^\alpha}(x) = E_u^\alpha(x, -b)$ for $b \in \{0, \dots, t/4\}$, $u \in [8]$ and $\alpha \in [m]$, denote the $3t/2$ -degree polynomial that packed-shares the coefficient vector $e_{(t/4+1) \cdot (u-1) + b}^\alpha$.
4. **Proof of product relation:** To check the product relation, each P_i requires to verify that $E_{-\ell}^\alpha(i) = A^\alpha(-\ell, i) \cdot B^\alpha(-\ell, i) - C^\alpha(-\ell, i)$ holds for each $\ell \in \{0, \dots, t/2\}$. Let $E^\alpha(i) = (E_{-t/2}^\alpha(i), \dots, E_0^\alpha(i))$ and $f^{E^\alpha(i)}(x)$ be the $3t/2$ -degree polynomial that packed-shares of $E^\alpha(i)$. Note that each $f^{E^\alpha(i)}(x)$ is a linear combination of the polynomials $(f^{e_j^\alpha}(x))_{j \in \{0, \dots, 2t\}}$ for each $\alpha \in [m]$.
 - (a) **Reconstructing $E^\alpha(i)$ and $E^\alpha(0)$ towards each P_i :** For each party P_i , ev-

ery party P_j computes $f^{E^\alpha(i)}(j) = \sum_{k=0}^{2t} j^k \cdot f_k^{e^\alpha}(j)$ and $f^{E^\alpha(0)}(j) = f_{e_0^\alpha}(j)$ for each $\alpha \in [m]$. Parties invoke $\mathcal{F}_{\text{privRec}}^{\text{batched}}$ (Functionality 4.7.1) where P_j inputs $(f^{E^\alpha(0)}(j), f^{E^\alpha(1)}(j), \dots, f^{E^\alpha(n)}(j))_{\alpha \in [m]}$.

- i. If the output is discard, then proceed to Step 5a.
- ii. If the output is (detect, Bad), then set $\text{ZEROS} = \text{ZEROS} \cup \text{Bad}$. If $|\text{ZEROS}| > t$ then go to Step 5a. Otherwise, go to Step 2.
- iii. Otherwise, P_i sets $f^{E^\alpha(0)}(x) = h_0^\alpha(x)$ and $f^{E^\alpha(i)}(x) = h_i^\alpha(x)$ for each $\alpha \in [m]$ such that $\mathcal{F}_{\text{privRec}}^{\text{batched}}$ did not output kill.

(b) **Verifying the product relation of each P_i and that $E^\alpha(0) = e_0^\alpha = (0, \dots, 0)$ holds:**

- i. Each P_i verifies that $E^\alpha(i)$ obtained from reconstructed polynomial $f^{E^\alpha(i)}(x)$ matches with $(f_i^{\alpha,a}(-\ell) \cdot f_i^{\alpha,b}(-\ell) - f_i^{\alpha,c}(-\ell))_{\ell \in \{0, \dots, t/2\}}$. P_i also verifies that $f^{E^\alpha(0)}(-\ell) = 0$ holds for each $\ell \in \{0, \dots, t/2\}$ and each $\alpha \in [m]$ for which $\mathcal{F}_{\text{privRec}}^{\text{batched}}$ did not output kill. If not, then P_i broadcasts complaint(i, β_i) where β_i is the smallest α for which the verification fails.
- ii. Parties construct a binary vector $(\text{pub}_1, \dots, \text{pub}_n)$ where $\text{pub}_i = 1$ if P_i broadcasted complaint(i, β_i).
- iii. Parties invoke $\mathcal{F}_{\text{pubRec}}$ (Functionality 4.6.5) five times where each P_j inputs $(\text{pub}_1, \dots, \text{pub}_n)$, ZEROS in every instance and the following shares respectively in the five instances: (i) $(f^{E^{\beta_1}(1)}(j), \dots, f^{E^{\beta_n}(n)}(j))$, (ii) $(f^{E^{\beta_1}(0)}(j), \dots, f^{E^{\beta_n}(0)}(j))$, (iii) $(g_j^{\beta_1,a}(1), \dots, g_j^{\beta_n,a}(n))$, (iv) $(g_j^{\beta_1,b}(1), \dots, g_j^{\beta_n,b}(n))$ and (v) $(g_j^{\beta_1,c}(1), \dots, g_j^{\beta_n,c}(n))$. For P_i which did not broadcast complaint(i, β_i), each P_j sets the i th share to be 0 in the five instances above. For any of the above instances:
 - A. If the output is discard, then proceed to Step 5a.
 - B. If the output is (detect, Bad), then set $\text{ZEROS} = \text{ZEROS} \cup \text{Bad}$. If $|\text{ZEROS}| > t$ then go to Step 5a. Otherwise, go to Step 2.
 - C. Otherwise, when the output is proceed, all parties set $f^{E^{\beta_i}(i)}(x), f^{E^{\beta_i}(0)}(x)$ and $f_i^{\beta_i,a}(x), f_i^{\beta_i,b}(x), f_i^{\beta_i,c}(x)$ as the respective output obtained in the corresponding five instances of $\mathcal{F}_{\text{pubRec}}$ for each P_i which broadcasted complaint(i, β_i).
- iv. For each P_i which complained, parties verify the checks as in Step 4(b)i using the polynomials returned by the instances of $\mathcal{F}_{\text{pubRec}}$. If it does not hold for some complaining party, then go to Step 5a. Otherwise, go to Step 5b.

5. Output:

- (a) **Discard:** All parties output \perp .
- (b) **Successful:** Each P_i outputs $(f_i^{\alpha,a}(x), g_i^{\alpha,a}(y), f_i^{\alpha,b}(x), g_i^{\alpha,b}(y), f_i^{\alpha,c}(x), g_i^{\alpha,c}(y))$ for each $\alpha \in [m]$ for which $\mathcal{F}_{\text{privRec}}^{\text{batched}}$ did not output kill in Step 4a. Here, $\{(f_i^{\alpha,a}(-\ell), f_i^{\alpha,b}(-\ell), f_i^{\alpha,c}(-\ell))\}_{\ell \in \{0, \dots, t/2\}}$ defines P_i 's shares of the multiplication triples.
-

Theorem 4.7.7. Protocol 4.7.6, $\Pi_{\text{PVTs}}^{\text{batched}}$, perfectly securely computes Functionality 4.7.5, $\mathcal{F}_{\text{PVTs}}^{\text{batched}}$, in the presence of a malicious adversary controlling at most $t < n/3$. It requires a communication complexity of $\mathcal{O}(mn^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast, and $\mathcal{O}(1)$ rounds. Each party broadcasts at most $\mathcal{O}(n \log n)$ bits.

Note that since (at most) n instances can be compromised (due to kill as output from $\mathcal{F}_{\text{privRec}}^{\text{batched}}$), the dealer shares $\mathcal{O}((m-n)n)$ multiplication triples. Consequently, we have that when $m-n = \mathcal{O}(n)$, the dealer generates $\mathcal{O}(n^2)$ triples at an amortized cost of $\mathcal{O}(n \log n)$ bits over point-to-point channels and $\mathcal{O}(1)$ bits of broadcast per triple.

4.8 Linear Perfectly Secure MPC

In this section, we first give details of the additional building blocks necessary for MPC such as reconstruction of degree- t polynomials, and Beaver triple generation. We conclude with our complete MPC protocol relying on these building blocks, the packed secret sharing (Sections 4.4, 4.5) and the verifiable triple sharing (Sections 4.6, 4.7). In the following, we use $\langle v \rangle$ to denote the degree- t Shamir-sharing of a value v among parties.

4.8.1 Secret Reconstruction

Since the outcome of our VSS is secrets in Shamir-shared format, we discuss how such sharing can be reconstructed efficiently. We use two standard ways of reconstruction:

Private reconstruction. Here, the secret is reconstructed privately to a specified party. This can be achieved by simply letting all the parties disclose the shares to the party who applies RS error correction for recovering the secret. We denote this protocol as Π_{Rec} . This requires $\mathcal{O}(n \log n)$ bits communication.

Batched public reconstruction. Naïvely, reconstructing $t+1$ secrets that are Shamir-shared requires $(t+1)n$ private reconstructions (via Π_{Rec}), resulting in $\mathcal{O}(n^3 \log n)$ communication¹.

¹Alternatively, $(t+1)n$ elements of broadcasts. Broadcasts are expensive and would require a minimum of $\mathcal{O}(n^3 \log n)$ communication and a minimum of constant expected inflation in the round complexity.

On the other hand the batch reconstruction protocol, first presented in [57], allows parties to robustly reconstruct $t + 1$ Shamir-shared values at a cost of communicating $\mathcal{O}(n^2 \log n)$ bits, ensuring an amortized cost of $\mathcal{O}(n \log n)$ bits per reconstruction.

In particular, given $\langle v_0 \rangle, \dots, \langle v_t \rangle$, parties translate them to n sharings *non-interactively*, say $\langle v'_1 \rangle, \dots, \langle v'_n \rangle$, using a linear error correcting code, such as Reed-Solomon code which tolerates up to t errors. To be specific, (v'_1, \dots, v'_n) can be thought of as n points on a t -degree polynomial $p(x) = \sum_{i=0}^t v_i x^i$. Following this, of the n sharings, one sharing $\langle v'_i \rangle$ is reconstructed towards each party P_i via private reconstruction protocol Π_{Rec} who obtains v'_i . At this stage, the parties essentially hold $\langle v_0 \rangle$. Therefore, n instances of private reconstruction enables every party to recover $p(x)$, the polynomial used to share v_0 , whose coefficients are the desired output. This requires a total communication of $\mathcal{O}(n^2 \log n)$ bits. The protocol Π_{bPubRec} appears below for completeness.

Protocol 4.8.1: Batched Public Reconstruction Protocol – Π_{bPubRec}

Common input: The description of a field \mathbb{F} , n non-zero distinct elements $1, \dots, n$.

Input: Parties hold the univariate degree- t sharings $\langle v_0 \rangle, \dots, \langle v_t \rangle$.

1. Let $p(x) = v_0 + v_1 x + v_2 x^2 + \dots + v_t x^t$.
 2. For each P_i , parties locally compute $\langle v'_i \rangle = \langle p(i) \rangle = \langle v_0 \rangle + \langle v_1 \rangle \cdot i + \langle v_2 \rangle \cdot i^2 + \dots + \langle v_t \rangle \cdot i^t$.
 3. For each party P_i , parties invoke Π_{Rec} with $\langle v'_i \rangle$ as input to enable P_i to privately reconstruct $v'_i = p(i)$. Note that parties now hold $\langle p(0) \rangle$.
 4. For each party P_i , parties invoke Π_{Rec} with $\langle p(0) \rangle$ as input to enable P_i to privately reconstruct the polynomial $p(x)$. Upon reconstructing, each P_i outputs the $t + 1$ coefficients v_0, v_1, \dots, v_t of $p(x)$.
-

Lemma 4.8.2. *Protocol 4.8.1, Π_{bPubRec} , has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and no broadcast for publicly reconstructing $\mathcal{O}(n)$ values (i.e., $\mathcal{O}(n \log n)$ bits) simultaneously in 2 rounds.*

4.8.2 From the PSS and VTS to MPC

We now give the road-map for our MPC. Excluding the PSS and the VTS, the existing tools are taken from [50]. Our protocol has two phases: (a) preparation of Beaver triples $(\langle a_l \rangle, \langle b_l \rangle, \langle c_l \rangle)_{l \in C}$, where C is the number of multiplication gates in circuit to be evaluated; (b) batched evaluation of the multiplication gates consuming the Beaver triples. Let us start with the latter.

Batched Beaver Multiplication. This protocol relies on the well known technique of Beaver’s circuit randomization [23], which, given a pre-computed t -shared random and private multiplication triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, reduces the computation of $\langle xy \rangle$ from $\langle x \rangle$ and $\langle y \rangle$ to two public reconstructions. Towards this, parties first locally compute $\langle d \rangle = \langle x \rangle - \langle a \rangle$ and $\langle e \rangle = \langle y \rangle - \langle b \rangle$, followed by public reconstruction of d and e . Since $z = xy = ((x - a) + a)((y - b) + b) = (d + a)(e + b) = de + db + ea + ab$, parties can locally compute $\langle z \rangle = \langle xy \rangle$ using the shared multiplication triple and the publicly reconstructed values d and e . Specifically, parties locally compute $\langle xy \rangle = de + d\langle b \rangle + e\langle a \rangle + \langle c \rangle$.

To leverage the efficiency benefits offered by the batch public reconstruction protocol, the protocol handles a batch of l multiplications together, each requiring 2 reconstructions. The $2l$ public reconstructions are thus batched together in groups of $t + 1$ to invoke Π_{bPubRec} and ensure an amortized communication complexity of $\mathcal{O}(n \log n)$ bits per reconstruction. The resultant communication complexity of Π_{bBeaver} for handling l multiplications is $\mathcal{O}((n^2 + nl) \log n)$. The formal description appears in Protocol 4.8.3.

Protocol 4.8.3: Batched Beaver Multiplication – Π_{bBeaver}

Input: Parties hold l degree- t shared triples $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$ for every $i \in [l]$ and l degree- t shared pairs of values $(\langle x_i \rangle, \langle y_i \rangle)$ to be multiplied.

1. For each $i \in [l]$, parties locally compute $\langle d_i \rangle = \langle x_i \rangle - \langle a_i \rangle$ and $\langle e_i \rangle = \langle y_i \rangle - \langle b_i \rangle$.
 2. Let $2l = k(t + 1)$. Parties execute k parallel instances of Π_{bPubRec} and publicly reconstruct $\{d_i, e_i\}$ for every $i \in [l]$.
 3. For each $i \in [l]$, parties locally compute $\langle z_i \rangle = \langle x_i y_i \rangle = d_i e_i + d_i \langle b_i \rangle + e_i \langle a_i \rangle + \langle c_i \rangle$.
-

Lemma 4.8.4. *Protocol 4.8.3, Π_{bBeaver} , has a communication complexity of $\mathcal{O}((ln + n^2) \log n)$ bits over point-to-point channels and no broadcast for the multiplication of l pairs of shared values in 2 rounds.*

Preparing Beaver triples. This task is further phrased in two tasks. First, a verifiable triple sharing (VTS) is used to make a dealer Shamir-share three values (a, b, c) such that $c = ab$. Second, a triple extraction protocol that takes n verified triples, i th one contributed by P_i and extracts $t/2$ triples that are unknown to the adversary.

Verifiable Triple Sharing. During this phase, each party shares verified multiplication triples which are subsequently consumed to extract random triples (unknown to any party) required

for circuit evaluation via Beaver’s trick [23]. Towards that, each party invokes $\mathcal{F}_{\text{PVT S}}^{\text{batched}}$ (Functionality 4.7.5) in parallel to generate the desired number of triples in a batched manner. The exact number of triples that a party has to share depends on the number of multiplication gates in the circuit, and we provide a cost analysis in Lemma 4.8.10. We thus have that there are n parallel instances of $\mathcal{F}_{\text{PVT S}}^{\text{batched}}$, where in each instance a party broadcasts $n \log n$ bits. Considering all the n instances, each party broadcasts $n^2 \log n$ in parallel. For this, we use the parallel broadcast primitive $\mathcal{F}_{\text{BC}}^{\text{parallel}}$ (Functionality 4.3.5). Thus, for the parallel batched verifiable triple sharing, we have the following.

Lemma 4.8.5. *Parallel batched verifiable triple sharing requires a communication of $\mathcal{O}(mn^3 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^3 \log n)$ bits broadcast in $\mathcal{O}(1)$ rounds. Each party broadcasts $\mathcal{O}(n^2 \log n)$ bits in parallel.*

Using the broadcast realisation of [7], we have that the parallel batched verifiable triple sharing requires $\mathcal{O}(mn^3 \log n + n^4 \log n)$ bits over point-to-point channels.

At the termination of this phase, we have n sets of triples, one shared by each party. Note that for each corrupt party which was discarded during $\mathcal{F}_{\text{PVT S}}^{\text{batched}}$, all parties assume default sharing of some publicly known triples. Given this, the next phase “merges” the triples shared by, and known to each party respectively to extract random triples.

Triple Extraction. Our last component is a triple extraction protocol that consumes one (verified) multiplication triple, say $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$, shared by each party P_i in the prior stage and extracts $\mathcal{O}(n)$ random triples not known to any party at the cost of $\mathcal{O}(n^2)$ point to point communication. In particular, the protocol extracts $h + 1 - t$ multiplication triples, where $h = \lfloor \frac{n-1}{2} \rfloor$ using n triples, one shared by each party. At a high level, the protocol proceeds as follows. First, the parties “transform” the n random shared triples $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$ for each $i \in [n]$ into n correlated triples $(\langle x_i \rangle, \langle y_i \rangle, \langle z_i \rangle)$ for every $i \in [n]$ such that the values $\{x_i, y_i, z_i\}_{i \in [n]}$ lie on the polynomials $X(\cdot), Y(\cdot)$ and $Z(\cdot)$ of degree h, h and $2h$ respectively where $X(\cdot) \cdot Y(\cdot) = Z(\cdot)$. Specifically, for each $i \in [n]$, it holds that $X(i) = x_i, Y(i) = y_i$ and $Z(i) = z_i$ where $1, \dots, n$ are publicly known distinct elements from \mathbb{F} . Furthermore, the transformation ensures that the adversary knows $\{x_i, y_i, z_i\}$ only if P_i is corrupt. This implies that the adversary may know t points on each of the polynomials $X(\cdot), Y(\cdot)$ and $Z(\cdot)$ of degree h, h and $2h$ respectively, thus guaranteeing a degree of freedom of $h + 1 - t = t/2$ in $X(\cdot), Y(\cdot)$ (and thus $Z(\cdot)$). Parties thus output the shared evaluation of these polynomials at $h + 1 - t$ publicly known points $\beta_1, \dots, \beta_{h+1-t}$ as the extracted shared multiplication triples.

The transformation itself works as follows. The parties simply set $x_i = a_i, y_i = b_i, z_i = c_i$ for $i \in \{1, \dots, h + 1\}$. Next, $\langle x_i \rangle$ and $\langle y_i \rangle$ for every $i \in \{h + 2, \dots, n\}$ can be computed

non-interactively by taking linear combination of $\{x_i, y_i\}_{i \in [h+1]}$. Following this, $\langle z_i \rangle$ for every $i \in \{h+2, \dots, n\}$ is computed using Beaver's trick where the inputs are $\langle x_i \rangle$ and $\langle y_i \rangle$ and the triple $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$. Clearly, if P_i is corrupt then x_i, y_i, z_i is known to the adversary as claimed. To conclude, we note that triple extraction reduces to running a batch of $\mathcal{O}(n)$ Beaver multiplications which requires $O(n^2 \log n)$ bits communication using Π_{bPubRec} . The formal description appears in Protocol 4.8.6.

Protocol 4.8.6: Triple Extraction – $\Pi_{\text{tripleExt}}$

Common input: The description of a field \mathbb{F} , $n = 2h + 1$ non-zero distinct elements $1, \dots, n$ and $h + 1 - t$ non-zero distinct elements $\beta_1, \dots, \beta_{h+1-t}$.

Input: Parties hold the degree- t shared triples $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$ for every $i \in [n]$ such that (a_i, b_i, c_i) is known to party P_i .

1. For each $i \in [h+1]$, parties locally set $\langle x_i \rangle = \langle a_i \rangle$, $\langle y_i \rangle = \langle b_i \rangle$ and $\langle z_i \rangle = \langle c_i \rangle$.
 2. Let $X(\cdot)$ and $Y(\cdot)$ be the degree- h polynomials defined by the points $\{x_i\}_{i \in [h+1]}$ and $\{y_i\}_{i \in [h+1]}$ respectively such that $X(i) = x_i$ and $Y(i) = y_i$ for all $i \in [h+1]$.
 3. For each $i \in \{h+2, \dots, n\}$, parties locally compute $\langle x_i \rangle = \langle X(i) \rangle$ and $\langle y_i \rangle = \langle Y(i) \rangle$.
 4. Parties invoke Π_{bBeaver} with $\{\langle x_i \rangle, \langle y_i \rangle, \langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle\}_{i \in \{h+2, \dots, n\}}$ and obtain $\{\langle z_i \rangle\}_{i \in \{h+2, \dots, n\}}$ where $z_i = x_i y_i$ for every $i \in \{h+2, \dots, n\}$.
 5. Let $Z(\cdot)$ be the degree- $2h$ polynomial defined by the points $\{z_i\}_{i \in [n]}$ such that $Z(i) = z_i$ for all $i \in [n]$.
 6. Parties locally compute $\langle \mathbf{a}_i \rangle = \langle X(\beta_i) \rangle$, $\langle \mathbf{b}_i \rangle = \langle Y(\beta_i) \rangle$ and $\langle \mathbf{c}_i \rangle = \langle Z(\beta_i) \rangle$ for every $i \in [h+1-t]$.
-

Lemma 4.8.7. Protocol 4.8.6, $\Pi_{\text{tripleExt}}$, has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and no broadcast for sharing $\mathcal{O}(n)$ random multiplication triples in 2 rounds.

4.8.3 The MPC Protocol

The protocol Π_{MPC} and the corresponding functionality \mathcal{F}_{MPC} are provided below. As described, at a high level, the protocol is divided into the following two phases:

1. *Beaver triple generation:* In this phase, parties generate C number of degree- t Shamir-shared multiplication triples where, C denotes the number of multiplication gates in the circuit. Towards that, each party first generates triples using our VTS protocol.

Subsequently, a triple extraction protocol “merges” the triples generated by all parties and “extracts” random triples (not known to any party) which will be consumed in the second phase. For sufficiently large circuits, specifically for circuits of size $\Omega(n^3)$, this phase incurs an amortized cost of $\mathcal{O}(n \log n)$ bits point-to-point communication per triple.

2. *Circuit computation:* Upon sharing of inputs by the input holding parties, in this phase the computation of the circuit proceeds by parties performing shared evaluation of the circuit. Since our sharing is linear, the linear operations of addition and multiplication by a constant are local. For multiplication of shared values, parties consume the Beaver triples generated in the prior phase. This is followed by the reconstruction of the outputs to the designated parties to complete the circuit evaluation.

Functionality 4.8.8: MPC – \mathcal{F}_{MPC}

Input: Each P_i holds input $x_i \in \mathbb{F} \cup \{\perp\}$.

Common Input: An n -party function $f(x_1, \dots, x_n)$.

1. Each P_i sends x_i to the functionality. For any P_i , if x_i is outside the domain or P_i did not send any input, set x_i to a predetermined default value.
 2. Compute $(y_1, \dots, y_n) = f(x_1, \dots, x_n)$ and send y_i to P_i for every $i \in [n]$.
-

Protocol 4.8.9: MPC – Π_{MPC}

Common input: The description of a circuit, the field \mathbb{F} , n non-zero distinct elements $1, \dots, n$ and a parameter h where $n = 2h + 1$. Let $m = \lceil \frac{C}{h+1-t} \rceil$.

Input: Parties hold their inputs (belonging to $\mathbb{F} \cup \{\perp\}$) to the circuit.

(Beaver triple generation:)

1. Each P_i chooses $m + n(t/2 + 1)$ random multiplication triples and executes $\Pi_{\text{PVT}}^{\text{batched}}$ (Section 4.7, Protocol 4.7.6) batching $\lceil \frac{m}{(t/2+1)} \rceil + n$ instances each with $t/2 + 1$ triples. Let $(\langle a_i^j \rangle, \langle b_i^j \rangle, \langle c_i^j \rangle)$ for $j \in [m]$ denote the triples shared by P_i .
2. Parties execute m instances of $\Pi_{\text{tripleExt}}$ (Protocol 4.8.6) with $(\langle a_i^j \rangle, \langle b_i^j \rangle, \langle c_i^j \rangle)$ for every $i \in [n]$ as the input for the j^{th} instance. Let $(\langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle)$ for $i \in [C]$ denote the random multiplication triples generated.

(Circuit computation:)

1. **(Input)** Each party P_i holding k_i inputs to the circuit executes $\Pi_{\text{pSS}}^{\text{batched}}$ (Section 4.5) batching $\lceil \frac{k_i}{t/2+1} \rceil$ instances to share its inputs.
 2. **(Linear Gates)** Parties locally apply the linear operation on their respective shares of the inputs.
 3. **(Multiplication Gates)** Let $(\langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle)$ be the multiplication triple associated with the i^{th} multiplication gate with shared inputs $(\langle x_i \rangle, \langle y_i \rangle)$. Parties invoke Π_{bBeaver} (Protocol 4.8.3) with $\{\langle x_i \rangle, \langle y_i \rangle, \langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle\}$ for all gates i at the same layer of the circuit and obtain the corresponding $\langle z_i \rangle$ as the output sharing for every gate i .
 4. **(Output)** For each output gate j with the associated sharing $\langle v_j \rangle$, parties execute Π_{Rec} towards every party P_i who is supposed to receive the output v_j .
-

Theorem 4.8.10. *Let $t < n/3$. Protocol 4.8.9 securely implements \mathcal{F}_{MPC} (Functionality 4.8.8) and has a communication complexity of $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$ bits over point to point channels and $\mathcal{O}(n^3 \log n)$ bits broadcast for evaluating a circuit with C gates and depth D in expected $\mathcal{O}(D)$ rounds. Every party broadcasts $\mathcal{O}(n^2 \log n)$ bits.*

Proof. The circuit evaluation requires C random multiplication triples. We analyse the cost of the two phases separately.

Beaver triple generation. Note that the triple extraction protocol ($\Pi_{\text{tripleExt}}$) generates $\mathcal{O}(n)$ (specifically, $h + 1 - t$ where $h = \lfloor \frac{n-1}{2} \rfloor$) such random triples by consuming one verified multiplication triple per party. Thus, we need $\mathcal{O}(C/n)$ instances of the triple extraction protocol, which incurs a cost of $\mathcal{O}(Cn \log n + n^2 \log n)$ bits over point-to-point channels. Each instance of triple extraction consumes one verified multiplication triple per party. This requires each party to ensure the sharing of a set of $\mathcal{O}(C/n)$ verified multiplication triples. Since our verified triple sharing packs $\mathcal{O}(n)$ (specifically, $t/2 + 1$) triples in one instance, this corresponds to each party parallelly running $\mathcal{O}(C/n^2) + n$ instances of packed verifiable triple sharing in a batched manner, where the additional n accounts for the (at most) n instances that the adversary can compromise in $\mathcal{F}_{\text{PVTs}}^{\text{batched}}$. This phase incurs a cost of $\mathcal{O}(Cn \log n + n^4 \log n)$ bits of communication over point-to-point channels and $\mathcal{O}(n^3 \log n)$ bits of broadcast, where every party broadcasts $\mathcal{O}(n^2 \log n)$ in parallel.

Circuit computation. In this phase, parties batched the multiplication gates at the same level in the circuit and invoke the batched Beaver multiplication protocol (Π_{bBeaver}) for evaluating them. Given C_i is the number of gates per level of the circuit, this stage incurs a cost of $\mathcal{O}(C_i \cdot n \log n + n^2 \log n)$ bits communication over point-to-point channels. Con-

sequently, we have that the circuit computation requires $\sum_{i=1}^D \mathcal{O}(C_i \cdot n \log n + n^2 \log n) = \mathcal{O}(Cn \log n + Dn^2 \log n)$ bits communication over point-to-point channels. \square

Chapter 5

Perfect Asynchronous MPC with Linear Communication Overhead

In this chapter, we elaborate on the MPC protocol with perfect security and optimal resilience in the asynchronous setting. In the process, we discuss our primary technical contribution, an improved protocol for asynchronous weak-binding secret sharing via the use of trivariate polynomials.

5.1 Introduction

We consider the most demanding setting: *perfect security with optimal resilience in the asynchronous model*. From the known lower bound of [30, 31, 8], perfect security implies that the number of corruptions in this setting is at most $t < n/4$, and hence the optimal resilience is when $n = 4t + 1$. This is in contrast to $n = 3t + 1$ in the synchronous setting. The seminal work of [30, 40] obtains perfect security with optimal resilience in the asynchronous model.

Communication Complexity of Asynchronous MPC

As described, the communication efficiency of MPC protocols is measured by the (amortized) cost of their communication complexity per multiplication gate. In the perfectly secure, optimally-resilient synchronous model, $\mathcal{O}(n \log n)$ communication complexity per multiplication gate was obtained nearly 15 years ago by the work of [26] (recently [9] improves the round complexity from $\mathcal{O}(D + n)$ to expected $\mathcal{O}(D)$ for circuits of depth D).

Progress in the (perfectly secure, optimally resilient) asynchronous model over the last 30 years has been slower. The work of [30] obtained $\tilde{\mathcal{O}}(n^6)$ per gate. [101, 95] improve to $\tilde{\mathcal{O}}(n^5)$ per gate. [25] improves to $\tilde{\mathcal{O}}(n^3)$ per gate. The best current bounds are by [92, 93] that obtained $\tilde{\mathcal{O}}(n^2)$ communication complexity per multiplication gate. The natural question

of obtaining $\tilde{\mathcal{O}}(n)$ complexity per multiplication gate for perfectly secure, optimally-resilient asynchronous protocols remained open for 30 years.

Our Main Result

Our main result is a perfectly secure, optimally-resilient asynchronous MPC protocol that achieves $\tilde{\mathcal{O}}(n)$ communication per multiplication gate.

Theorem 5.1.1 (Main Result). *For a circuit with C multiplication gates and depth D there exists a perfectly-secure, optimally-resilient asynchronous MPC protocol with $\mathcal{O}((Cn + Dn^2 + n^5) \log n)$ communication complexity and $\mathcal{O}(D)$ expected run-time.*

Previously, the best-known result required $\mathcal{O}((Cn^2 + Dn^2 + n^5) \log n)$ communication complexity [92].

Main Technical Result

Our main technical result is a new Asynchronous Weak-Binding Secret Sharing that costs just $\mathcal{O}(\log n)$ bits of communication complexity per secret of size $\log n$ bits. It is perfectly secure, resilient to $t < n/4$, and has constant round complexity and polynomial computation complexity. For our MPC purposes, we do not need the reconstruction of those shares; we just need the dealer to commit to a well-defined polynomial. We call this property as “weak-binding”.

Theorem 5.1.2 (Asynchronous Weak-binding Secret Sharing (informal)). *There exists a perfectly secure, optimally resilient protocol for asynchronous weak-binding secret sharing that can share $\mathcal{O}(n^4)$ secrets in constant time, with communication complexity of $\mathcal{O}(\log n)$ bits per secret of size $\log n$ bits.*

Many forms of verifiable secret sharing, both in the synchronous and asynchronous settings, rely on bivariate polynomial sharing. Our protocol is based on *trivariate polynomial* sharing. Our use of trivariate polynomial sharing approach follows the recent work of Appelbaum and Kachlon [15]. Nevertheless, we show how to reconstruct the trivariate polynomial for future reference and for independent interest. This variant is “weak” in the sense that reconstruction might fail (or not terminate). In that case, however, the honest dealer can shun a set of $t/2 + 1$ corrupted parties. Nonetheless, we remark again that we do not use reconstruction, and in particular, our final MPC construction does not shun parties and does not use player elimination.

The first asynchronous verifiable secret sharing protocol [27] achieves $\mathcal{O}(n^4)$ (amortized) overhead per secret. In comparison, our trivariate-based asynchronous weak-binding secret

sharing achieves $\mathcal{O}(1)$ (amortized) overhead per secret. Nevertheless, we remark again that our primitive is weaker as it does not guarantee reconstruction. Despite this fact, we show that this weak primitive suffices for the crux of our MPC, which is a distributed ZK proof of multiplication triplets. The aim of the ZK proof is to prove that some (secret) polynomial $p(x)$ possesses a certain degree. To accomplish this, the prover incorporates the coefficients of $p(x)$ as secrets in the trivariate polynomial and distributes shares on this trivariate polynomial. Since the trivariate polynomial can contain a predetermined number of secrets, the mere success of the sharing process and the existence of a well-defined trivariate polynomial are sufficient evidence to confirm that $p(x)$ indeed has the desired degree. Here, there is no need for reconstruction. Reconstruction would also reveal the coefficients of the polynomial $p(x)$, which have to remain secret.

5.1.1 Related work

In the setting of perfect security with a synchronous network, the work of [26, 76] achieved $\tilde{\mathcal{O}}(n)$ communication per multiplication gate. A lower bound of $\tilde{\mathcal{O}}(n)$ was later established in [58] for a resilience of $t < n/3$ which is known to be necessary in this setting. The recent work of [9] improves the round complexity of [26, 76] from $\mathcal{O}(D+n)$ to $\mathcal{O}(D)$ in expectation while maintaining linear communication complexity in the number of parties.

The results in the perfect asynchronous setting have been mentioned earlier and we avoid repetition here. We simply summarize that there is no linear overhead protocol in this setting thus far. Nonetheless, linear-overhead protocols have been achieved earlier in two weaker settings— (a) statistical security with non-optimal resilience of $t < n/4$ over asynchronous networks¹ [50] (b) perfect security with $t < n/4$ over hybrid network where the network permits a single synchronous round before turning to fully asynchronous mode [52, 50].

As mentioned, our trivariate secret sharing protocol is inspired by the work of Applebaum and Kachlon [15]. This work uses trivariate polynomial for constructing error-correcting code with quasipolynomial-time conflict-decoder.

5.2 Technical Overview

In this section, we provide a technical overview of our work. We give some background on basic asynchronous verifiable secret sharing (basically covering previous work) in Section 5.2.1, and proceed to our asynchronous weak-binding secret sharing in Section 5.2.2. We overview our MPC protocol in Section 5.2.3. Lastly, we conclude our triple secret sharing protocol in Section 5.2.4 which acts as the building block for MPC and builds upon our asynchronous

¹The optimal resilience for statistical asynchronous MPC is $t < n/3$ [31].

weak-binding secret sharing.

The model. Before we proceed, let us first introduce the model. We assume asynchronous communication, which means the adversary can arbitrarily delay messages sent between honest parties. However, such messages are eventually received. It is important to note that the adversary does not see the content of the messages (as we assume ideal channels between the honest parties), but it can see the type of messages that are being sent (e.g., identifying whether it's the first, second, or third message of the protocol). Since the adversary controls the corrupted parties, messages that are supposed to be sent by the corrupted parties to the honest parties might never be sent. Honest parties cannot distinguish whether a message is merely delayed or has not been sent altogether. Consequently, honest parties must continue waiting, with the potential consequence of certain foundational processes never reaching completion. However, it's important to highlight that the complete MPC protocol guarantees termination. This means that it possesses mechanisms to recognize non-terminating sub-protocols and take appropriate measures to bring them to a halt.

Besides the point-to-point channels, we assume for now the existence of a broadcast channel with the guarantee that (1) If the sender is honest and broadcasts M then eventually all honest parties will receive M ; (2) If some honest party received a message M (in an instance of a corrupted sender), then eventually all honest parties will receive M . This can be implemented by asynchronous broadcast or A-cast primitive [37]. The cost is $\mathcal{O}(n^2|M|)$ for broadcasting the message M .

5.2.1 Basic Asynchronous Verifiable Secret Sharing

Our starting point is a variant of the verifiable secret sharing due to Ben-Or, Canetti, and Goldreich [30]. In asynchronous verifiable secret sharing (AVSS), the dealer holds some secret s , and its goal is to distribute the shares to the parties. The parties then verify that the shares define a unique secret. At a later point, the parties might reconstruct the secret s . The properties that the AVSS offers are:

- **Validity:** If the dealer is honest, then the protocol must terminate. At the end of the reconstruction phase, all honest parties output s , the input of the dealer in the sharing phase;
- **Secrecy:** The view of the adversary in the sharing phase in the case of an honest dealer is independent of s ;
- **Binding:** The view of the honest parties at the end of the sharing phase (if terminated) uniquely defines some secret s' .

For simplicity, we assume for now that the dealer can efficiently solve the problem of finding the maximal clique in a graph. At a high level, the secret-sharing protocol proceeds as follows:

1. The dealer: Choose a random bivariate polynomial $S(\mathbf{x}, \mathbf{y})$ of degree- t in both variables such that $S(0, 0) = s$. Send over the private channels to each party P_i its shares $(S(\mathbf{x}, i), S(i, \mathbf{y}))$.
2. Each party P_i : Upon receiving the shares $(f_i(\mathbf{x}), g_i(\mathbf{y}))$ from the dealer, send to P_j the sub-shares $(f_i(j), g_i(j))$.
3. Each party P_i : Upon receiving $(u_{j,i}, v_{j,i})$ from party P_j , verify that $u_{j,i} = g_i(j)$ ($= f_j(i)$) and $v_{j,i} = f_i(j)$ ($= g_j(i)$). If so, then P_i broadcasts $\text{Good}(i, j)$.
4. The dealer: Initialize an undirected graph G over the vertices $V = [n]$. Upon seeing $\text{Good}(i, j)$ broadcasted by P_i and $\text{Good}(j, i)$ broadcasted by P_j , add the edge (i, j) to the graph. If a clique $K \subseteq [n]$ of cardinality at least $3t + 1$ is found in G , then broadcast (Clique, K) .
5. Each party P_i : Initialize a similar graph as the dealer in the previous step. Upon seeing a broadcasted message (Clique, K) from the dealer, verify that K is a $3t + 1$ clique in the graph. If not, continue to listen to Good messages broadcast and update the graph. Once K is verified:
 - (a) If $i \in K$, then halt and output $(f_i(x), g_i(y))$.
 - (b) If $i \notin K$, then wait to receive all sub-shares from parties in K (received from Step 3), and reconstruct the polynomials $f_i(x), g_i(y)$ using Reed-Solomon decoding.

We do not specify the reconstruction phase, as it is immediate and less relevant to our discussion. Moreover, note that the protocol might never terminate in a case of a corrupted dealer. E.g., the parties might wait forever for the dealer to broadcast the message (Clique, K) . A party cannot decide whether to abort or whether this message will eventually arrive.

We first claim that if one honest party terminates, all honest parties eventually terminate. An honest party terminates only after the dealer has broadcasted a clique K , and it validated that the clique exists in its graph. Since those are all broadcasted messages if one honest party saw this, all honest parties would eventually see the same property.

If a clique of $3t + 1$ parties is found, the clique must contain at least $2t + 1$ honest parties. All the messages of those honest parties are broadcasted; therefore, we know that their shares agree. Their shares define a unique bivariate polynomial $S(\mathbf{x}, \mathbf{y})$ of degree- t in both variables. It is also guaranteed that all honest parties eventually output shares on the same bivariate polynomial. Specifically, when running the Reed-Solomon decoding on messages received

from parties in K , there are $2t + 1$ shares on the polynomial $S(x, y)$ and at most t errors. Reed Solomon decoding results in shares on the bivariate polynomial. Finally, validity holds from the fact that when the dealer is honest, all honest parties agree with each other, and therefore a $3t + 1$ -clique must appear in the graph.

Making it polynomial time – the STAR algorithm. The problem with the above protocol is that the dealer has to solve clique, which is an NP-hard problem. A beautiful idea in the work of Ben-Or, Canetti, and Goldreich [30] (credit within is given to Canetti’s thesis [40]) shows that an *approximation* of clique suffices to bind a unique bivariate polynomial. Specifically, the dealer searches for a (C, D) -star, which is defined as follows:

(C, D)-Star: sets $C, D \subseteq [n]$ are Star in G if (1): $C \subseteq D$; (2) $|C| \geq 2t + 1$ and $|D| \geq n - t$; (3) For every $c \in C$ and $d \in D$ it holds that $(c, d) \in G$.

Note that C is a clique, whereas nodes in D agree with all nodes in C , but not necessarily with each other. The main idea is that if there exists a clique K of size $n - t \geq 3t + 1$ in G , it might be hard to find it, but it is easy to find smaller cliques of size $n - 2t \geq 2t + 1$. For example, to find such a clique, look at the complement graph \overline{G} . The clique K is now an independent set; Find a maximal matching in the graph \overline{G} ; let M be that maximal matching. The set $[n] \setminus M$ is an independent set in \overline{G} and, therefore, a clique in G . Moreover, when the dealer is honest, since all the edges are between honest parties and corrupted parties, or between corrupted parties, then M is of size at most $2t$. Therefore, $[n] \setminus M$ is of size $n - 2t$ and is a clique in the graph G . Canetti [40] shows a procedure that, if a $3t + 1$ clique exists, then it efficiently finds a (C, D) -STAR in a graph – i.e., a smaller clique (C) of size $2t + 1$, together with a larger set D where each $d \in D$ is connected to all of C .

The verifiable secret sharing is slightly more involved when the dealer finds a (C, D) -star and not a $3t + 1$ -clique. We do not get into the exact details. Yet, the main ideas why the STAR structure suffices are as follows:

- **Validity:** When the dealer is honest, then the dealer must eventually find a STAR. That is, when the dealer is honest, then eventually, we will have a clique K of size $3t + 1$ in the graph. In that case, the STAR algorithm always finds a (C, D) -star.
- **Binding:** If a (C, D) -star was found (either when the dealer is honest or corrupted), then a unique bivariate polynomial is defined from the shares of the honest parties in C . Since the size of C is at least $2t + 1$, it contains at least $t + 1$ honest parties. The shares of those honest parties uniquely define a bivariate polynomial. Moreover, the honest parties in D agree with all the honest parties in C ; therefore, their shares lie on

the same bivariate polynomial. At this point, we have at least $2t + 1$ honest parties that hold correct shares, and therefore all honest parties can eventually reconstruct correct shares.

Communication complexity. Before we proceed, let us first elaborate on the communication complexity of the protocol above. It is easy to see that the parties exchange a total of $\mathcal{O}(n^2 \log n)$ bits over the point-to-point channels and additional $\mathcal{O}(n^2 \log n)$ bits over the broadcast channel. This is translated to $\mathcal{O}(n^4 \log n)$ total communication complexity over point-to-point channels and no broadcast (using the broadcast protocol of [37]). That is, we have an overhead of $\mathcal{O}(n^4 \log n)$ for sharing just a single value!

5.2.2 Our Asynchronous Weak-Binding Secret Sharing

Our goal: $\mathcal{O}(1)$ overhead per secret. To achieve secure computation with linear communication complexity, our ultimate goal is to reach $\mathcal{O}(1)$ overhead per secret. This necessitates a substantial enhancement of the basic scheme by a factor of $\mathcal{O}(n^4)$. Borrowing ideas from the synchronous MPC [9], this goal is achieved via two routes: (1) batching; and (2) packing. However, packing in the asynchronous case gets an intriguing turn and requires borrowing new ideas.

Reducing the communication complexity by batching. It is immediate to reduce the communication complexity using a batching technique: The dealer will invoke m instances of AVSS in parallel. At the same time, the broadcasted information will be shared for all m instances. That is, a party P_i will broadcast $\text{Good}(i, j)$ only after it receives from P_j the subshares in all m instances and verifies that they agree with the share it received from the dealer in each one of the m instances. This reduces the total cost to $\mathcal{O}(m \cdot n^2 \log n)$ over point-to-point plus $\mathcal{O}(n^2 \log n)$ broadcast (notice that the broadcast cost is independent of m). Setting $m = n^2$, we obtain a protocol that runs in total communication complexity of $\mathcal{O}(n^4 \log n)$ bits for sharing n^2 secrets (each is of size $\log n$ bits). At this point, we have an overhead of $\mathcal{O}(n^2)$ per secret.

Reducing the communication complexity by packing. Packing in the asynchronous case turns out to be radically different than in the synchronous case. The main reason for the $\mathcal{O}(n^2)$ overhead in the secret sharing is because a single secret is hidden in a structure of size $\mathcal{O}(n^2)$, i.e., a bivariate polynomial. To achieve $\mathcal{O}(1)$ overhead, we have to pack $\mathcal{O}(n^2)$ secrets in a single bivariate polynomial or use a different structure.

In the basic AVSS protocol, a single secret is shared using a (t, t) -bivariate polynomial.¹ The adversary, which controls at most t parties, receives $f_i(\mathbf{x}) = S(\mathbf{x}, i)$ and $g_i(\mathbf{y}) = S(i, \mathbf{y})$ for every $i \in I$, when I is the set of corrupted parties, and assume for simplicity that $|I| = t$ (and not smaller). The f -shares give the adversary a total of $t(t+1)$ points on the polynomial. Since $f_i(k) = g_k(i)$ for every $i, k \in I$, the g -shares gives the adversary just additional $t(t+1) - t^2 = t$ “new” points on the polynomial (those are $g_i(0)$ for $i \in I$), and thus the adversary gets a total of $t^2 + 2t$. The polynomial $S(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^t \sum_{j=0}^t a_{i,j} x^i y^j$ contains a total of $(t+1)^2$ points, and thus we have just a single degree of freedom, i.e., we can hide just a single secret.

The main idea of packing is to use a polynomial of a higher degree while maintaining the same cost for the sharing and verification:

Using $(t + t/2) \times t$ -bivariate polynomial.² If we use a bivariate polynomial of degree, e.g., $(t + t/2, t)$, then there are in total $(t + t/2 + 1)(t + 1)$ values on the polynomial. The important part of this particular choice of parameters is that the degree of the y is t , which still allows using Reed-Solomon decoding as part of the protocol. Similar calculation as in the (t, t) case leads to the adversary’s shares revealing a total of $t(t + t/2 + 1) + t$ values. The bivariate polynomial, therefore, contains $t/2 + 1$ degrees of freedom, i.e., we can hide $\mathcal{O}(n)$ values in a single bivariate polynomial.

Packing $\mathcal{O}(n)$ values instead of $\mathcal{O}(1)$ reduced the overhead per secret from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. The important message here is that the STAR algorithm still suffices. Specifically, a (C, D) -star defines a unique bivariate polynomial: Recall that C is a clique of $2t + 1$ parties, and therefore it must contain at least $t + 1$ honest parties with their shares agreeing with each other. Moreover, all the honest parties in C are consistent with all the parties in D where D contains at least $2t + 1$ honest parties. The shares f shares of honest parties in C together with the g -shares of honest parties in D define a unique bivariate polynomial $S(\mathbf{x}, \mathbf{y})$ that satisfies $S(\mathbf{x}, c) = f_c(\mathbf{x})$ for every $c \in C$ (recall that each f_c has degree $t + t/2$). Moreover $S(d, \mathbf{y}) = g_d(\mathbf{y})$ holds for every honest party $d \in D$. Guaranteeing these parties will also hold correct f -shares requires some additional work, which was already shown in previous works (see [50]).

This protocol plays a pivotal role in our final construction, but for our ultimate goal, we still need to go one step further and push for $\mathcal{O}(1)$ -word overhead per secret. Note, however, that this one step forward will not give us verifiable secret sharing, but only some weaker

¹We use (q, ℓ) -bivariate polynomial to denote a bivariate polynomial which is of degree at most q in \mathbf{x} and at most ℓ in \mathbf{y} .

²We use $t + t/2$ just as an example. The above works for any $t \times (t + d)$ -bivariate polynomial for $d \leq t$ and $d \in \mathcal{O}(n)$.

form of sharing.

Using $(t + t/2) \times (t + t/2)$ -bivariate polynomial. If we use a bivariate polynomial of degree $(t + t/2, t + t/2)$, there are $(t + t/2 + 1)^2$ total values on the polynomial. Similar calculations as above lead to $(t/2 + 1)^2$ values that we can pack in the polynomial, i.e., $\mathcal{O}(n^2)$ secrets.

However, here the STAR technique does not give us a binding guarantee. A (C, D) -star provides a clique C of size $2t + 1$, which contains, in the worst case, $t + 1$ honest parties. Their f -shares, their g -shares, separately or combined, do not uniquely define a $(t + t/2) \times (t + t/2)$ -bivariate polynomial. Since the parties in D do not necessarily agree with each other, we cannot use their shares to define the bivariate polynomial before we have a unique one that is defined by the parties in C .

It is easy to see that an alternative, stronger guarantee, would suffice for binding:

(C, D)-BigStar: sets $C, D \subseteq [n]$ are BigStar in G if: (1) $C \subseteq D$; (2) $|C| \geq 2t + t/2 + 1$ and $|D| \geq n - t$; (3) For every $c \in C$ and $d \in D$ it holds that $(c, d) \in G$.

Note that the only difference between BigStar and Star is that C is of size $2t + t/2 + 1$ instead of $2t + 1$ as in Star. Such a larger set C ensures $t + t/2 + 1$ honest parties that agree with each other, hence defining a unique bivariate polynomial. However, we are unaware of any polynomial-time algorithm that finds BigStar in a graph. At this point, we can potentially reach the $\mathcal{O}(1)$ overhead in communication complexity, but at the expense of having the dealer find a large clique in exponential time. This is clearly not ideal.

Exponential-time improvement using trivariate polynomials. To solve the above problem, we add one more dimension, which will allow us, in particular, to pack $\mathcal{O}(n^3)$ secrets and find a BigStar efficiently, if it exists. Instead of using a bivariate polynomial $S(x, y)$ of degree $t + t/2$ in both variables x, y , the dealer will use a *trivariate* polynomial $S(x, y, z)$ of degree $t + t/2$ in all three variables x, y, z . We then naturally extend the protocol to trivariate sharing. E.g., the share of each party P_i is now *three* bivariate polynomials:

$$S(x, y, i), \quad S(x, i, z), \quad S(i, y, z) .$$

Two parties, P_i and P_j then exchange *six* univariate polynomials:

$$\begin{aligned} S(x, j, i), \quad S(x, i, j), \quad S(i, y, j) , \\ S(j, y, i), \quad S(j, i, z), \quad S(i, j, z) . \end{aligned}$$

and a party P_i broadcasts $\text{Good}(i, j)$ if the shares it receives from P_j (i.e., the six univariate polynomials) agree with those it received from the dealer.

The dealer then constructs a graph G as in the basic AVSS scheme. However, as mentioned, we are now looking for a more robust condition on the graph since our polynomials are of degree $t + t/2$. Thus, we need a clique of size $2t + t/2 + 1$, which will imply having at least $t + t/2 + 1$ honest parties. BigStar now also suffices for binding, but it is unclear how to find it just as in the bivariate sharing case. However, at this point, some seemingly weaker property on the graph also suffices to achieve binding, which was insufficient in bivariate sharing. The property is:

Dense: A set of vertices $L \subseteq [n]$ is called Dense if it is of cardinality at least $3t + 1$,¹ and each node in L has at least degree $3t + t/2 + 1$.

This is clearly a property that is easy to find in a graph G . The intuition is that the set L contains at least $2t + 1$ honest parties. Moreover, two honest parties P_k, P_ℓ that have degree $3t + t/2 + 1$ must agree with each other, even though they did not necessarily hear the shares of one another due to communication delays (we will see why this holds soon). The honest parties in Dense therefore define a clique of at least $2t + 1$ honest parties, and their shares define a unique trivariate polynomial.

Binding: For binding, it suffices to have one of the two following properties:

1. Dense: which essentially defines a clique of $2t + 1$ honest parties;
2. BigStar: where $|C| \geq 2t + t/2 + 1$, and $|D| \geq n - t$. The set C defines a clique of $t + t/2 + 1$ honest parties that agree with each other, which defines a unique trivariate polynomial. Moreover, the set D defines overall $2t + 1$ honest parties that have correct shares.

Therefore, whenever one of those properties occurs in the graph, we are satisfied and can terminate the protocol. We show a poly-time algorithm that finds those properties.

Validity. For validity, we have to guarantee that when the dealer is honest, it must find one of these two properties (Dense or BigStar) in the graph. In the honest dealer case, the graph will eventually contain a clique of size $3t + 1$. At this point, the following algorithm must find either Dense or BigStar:

1. The dealer looks for a Dense set L in the graph. If found, output (Dense, L).

¹We note that an L of size $2t + t/2 + 1$ would, in fact, suffice for defining a unique trivariate polynomial. However, we are able to find a larger set of size $3t + 1$ in polynomial time.

2. If the graph does not contain a dense set L , then it implies that there is at least one honest party, say P_j , whose degree is less than $3t + t/2 + 1$. Moreover, all missing edges in the graph are between honest parties and corrupted parties, or between corrupted parties. We then consider the graph G while considering only the neighbors of P_j (including P_j). We will obtain a graph of size $n' = 3t + t/2 + 1$ vertices, where all removed vertices correspond to corrupted parties, and so we have $t' = t/2$ corrupted parties remaining. Moreover, this graph contains a clique of size $3t + 1$. The standard (C, D) -star returns a clique $|C| \geq n' - 2t' \geq 2t + t/2 + 1$, and a set $|D| \geq n' - t' \geq 3t + 1$, which is essentially a BigStar in G . The dealer does the above for every low-degree party P_j until it hits on a BigStar, and this procedure is efficient.

Therefore, once we have a clique of $3t + 1$ honest parties, the dealer is guaranteed to find either a Dense or a BigStar in polynomial time.

Why do bivariate polynomials not suffice? A natural question is why we could not find the same property on the graph with bivariate polynomials, and we have to work with trivariates. This is an idea we borrow from [15], and is the crux of this part of the work. The property we need here is *transitivity*. Suppose P_k and P_j are both honest, and they agree with a common set E of at least $2t + 1$ honest parties, but they did not hear the sub-shares from each other since the adversary delays their communication. Do their shares agree?

Note that in the setting of the Dense graph, P_k and P_j are two honest parties that have a high degree of $3t + t/2 + 1$. This means they have at least $3t + 1$ parties in their intersection, which implies that they agree with some common set E of at least $2t + 1$ honest parties. To see whether or not their shares agree, let's consider bivariate sharing versus trivariate sharing:

1. **Bivariate:** In bivariate sharing, the parties hold univariate polynomials as shares, and they exchange points. It is easy to see that the shares of P_k and P_j do not necessarily agree with each other, even if they agree with a common set E of cardinality $2t + 1$. We can set P_k to have $f_k(\mathbf{x}), g_k(\mathbf{y})$ and P_j to have $f_j(\mathbf{x}), g_j(\mathbf{y})$ such that $f_k(j) \neq g_j(k)$ and $f_j(k) \neq g_k(j)$. Yet, we can give a set E of cardinality $2t + 1$ arbitrary shares such that $f_j(e) = g_e(j)$ and $g_j(e) = f_e(j)$ for every $e \in E$, and likewise, $f_k(e) = g_e(k)$ and $g_k(e) = f_e(k)$ for every $e \in E$. This does not impose many constraints on the polynomials $f_e(\mathbf{x}), g_e(\mathbf{y})$ for every $e \in E$. Therefore, the existence of the property Dense does not necessarily imply a clique of honest parties of large cardinality that agree with each other.
2. **Trivariate:** In trivariate sharing, the parties hold bivariate polynomials as shares, and they exchange univariate polynomials. If P_j and P_k have a common set of neighbors E

of cardinality $2t + 1$, then *they necessarily hold shares that agree with each other*¹. The key idea is that P_k exchanges with each party P_e for $e \in E$ univariate polynomials. Since the univariate polynomials agree, they also agree on the index j . The $2t + 1$ points of parties in E on the index j uniquely define the univariate polynomials that P_k expects to receive from P_j . Thus, even though P_k did not hear yet the message from P_j , and P_j is honest, it knows that the shares would agree, even if the dealer is corrupted. A similar argument holds for P_j . The formal argument appears in Section 5.3.

To conclude, to share the trivariate polynomial, we have a total of $\mathcal{O}(n^3 \log n)$ bits over point-to-point channel together with $\mathcal{O}(n^2 \log n)$ bits over the broadcast channel. If we batch n instances together, we get a total of $\mathcal{O}(n^4 \log n)$ bits over point-to-point channels and no broadcast for sharing $\mathcal{O}(n^4)$ secrets, each of $\log n$ bits. I.e., we obtain an overhead of $\mathcal{O}(1)$.

Note, however, that the sharing is weak. Namely, we know that there is a well-defined trivariate polynomial, but we cannot necessarily reconstruct it robustly. In particular, even in the case of an honest dealer, reconstruction might fail. In that case, however, the dealer can shun at least $t/2 + 1$ corrupted parties. We show the reconstruction in Section 5.6.

However, as we will see, the reconstruction is not required for our MPC, and the sharing itself suffices. We gave it for completeness and as it might be useful as an independent primitive.

5.2.3 Our MPC Protocol

Our MPC protocol follows the following structure: an offline phase in which the parties generate Beaver triplets [23], and an online phase in which the parties compute the circuit while consuming those triples.

Beaver triplets generation. Our goal is to distribute (Shamir, univariate degree- t) shares of random secret values a, b , and c , such that $c = ab$. If the circuit contains C multiplication gates, we need C such triplets. Towards that end, we follow the same steps as in [50], and generate such triplets in three stages:

1. **Triplets with a dealer:** Each party generates shares of a_i, b_i, c_i such that $c_i = a_i \cdot b_i$. We generate all the triplets in parallel using expected $\mathcal{O}(1)$ rounds. We overview this step below in Section 5.2.4. Our main contribution is in improving this step using the asynchronous weak-binding secret sharing with $\mathcal{O}(1)$ -overhead. Despite the fact that this secret sharing with $\mathcal{O}(1)$ -overhead is not robust, it suffices since it is used as part

¹In fact, as shown in [15], having a common set of honest neighbors of size $t + t/2 + 1$ suffices for P_j and P_k to hold shares that are consistent with the same trivariate polynomial of degree $t + t/2$ in each variable.

of a perfect zero-knowledge proof where the dealer proves that the shares it generated indeed correspond to a product relation. If the sharing fails, then we can simply ignore the contribution of that dealer.

In our protocol, each party acts as a dealer to generate C/n triplets. This step requires a cost of $\mathcal{O}(n^4 \log n + C \log n)$ communication for a single party and an overall cost of $\mathcal{O}(n^5 \log n + Cn \log n)$ for all the parties together.

2. **Agreeing on a core set (ACS):** All triplet generations of honest dealers eventually terminate, whereas those of corrupted dealers might never terminate. However, if one honest party sees that the triplet generation of some player P_i terminates, then all the honest parties will eventually receive output in that triplet generation.

As t instances might never terminate, the parties proceed with the protocol once at least $n - t$ parties successfully complete the triplet generation. Since parties might receive messages in different orders, they have to agree on the set of parties for which their triplet generation was successful.

The set Core of at least $n - t$ parties (who have successfully completed the triple sharing) will be chosen using the agreement on core set (ACS) protocol. The communication cost of ACS from [40] is $\mathcal{O}(n^7 \log n)$, and its run-time is $\log n$. In [10], the communication is improved to $\mathcal{O}(n^5 \log n)$, and run-time is expected constant time. We use the latter to quote our complexity. ¹

3. **Triplets with no dealer:** Once agreed which triplets to consider (Core), using triplet extraction of [50], we can extract from a total of $\frac{C(n-t)}{n}$ triplets generated by the dealers in Core, $\mathcal{O}(C)$ triplets where no party knows the underlying values. This step costs $\mathcal{O}(n^2 \log n + Cn \log n)$.

In summary, for generating C triplets we pay a total of $\mathcal{O}(n^5 \log n + Cn \log n)$.

The MPC protocol follows the standard structure where each party shares its input, and the parties evaluate the circuit gate-by-gate, or more exactly, layer-by-layer. In each multiplication gate, the parties have to consume one multiplication triple. Using the method of [50], if the i th layer of the circuit contains C_i multiplications (for $i \in [D]$, where D is the depth of the circuit), the evaluation costs $\mathcal{O}(n^2 \log n + C_i \cdot n \log n)$. Summing over all layers, this is $\sum_{i \in [D]} (n^2 + nC_i) \log n = (Dn^2 + Cn) \log n$. Together with the generation of the triplets, we get the claimed $\mathcal{O}((Cn + Dn^2 + n^5) \log n)$ cost as in Theorem 5.1.1. We refer the reader to Section 5.8 for further details on our MPC protocol.

One point worth addressing is that we parallelize the input-sharing phase to the triplet

¹Without [10], the cost of our entire MPC is $\mathcal{O}((Cn + Dn^2 + n^7) \log n)$ and $\mathcal{O}(D + \log n)$ expected-time.

generation. In that case, the ACS protocol selects the parties for which their triplet generation and input sharing were successful. Moreover, the input-sharing phase uses the robust secret-sharing with $\mathcal{O}(n)$ overhead and not the non-robust $\mathcal{O}(1)$. After the parties obtain robust shares (either sharing inputs or of product relations), all other computations are merely reconstructions and linear combinations of shared values. Since messages of honest parties are guaranteed to be delivered, and we have at least $2t + 1$ honest parties in Core with at most t corruptions, all reconstructions are guaranteed to terminate successfully and asynchrony has no effect.

5.2.4 Multiplication Triplets with a Dealer

The goal is that a dealer wishes to distribute shares of secret values $\vec{a}, \vec{b}, \vec{c}$ such that for every i it holds that $c_i = a_i b_i$. Towards this end, the dealer plants \vec{a} into some bivariate polynomial $A(\mathbf{x}, \mathbf{y})$ using the asynchronous VSS scheme that employs a bivariate polynomial of degree- $(t+t/2, t)$. Such a VSS is given in [50], which we slightly simplify and give it for completeness in Section 5.4. Specifically, \vec{a} is placed at $(A(-\beta, 0))_{\beta \in \{0, \dots, t/2\}}$. Similarly, the dealer plants \vec{b} into $B(\mathbf{x}, \mathbf{y})$ and \vec{c} into $C(\mathbf{x}, \mathbf{y})$. It is important to note that we deploy robust AVSS here, to ensure that the triplets are shared via degree- t polynomials (which is utilized by our MPC protocol). So we can plant only $\mathcal{O}(n)$ values in each bivariate polynomial. Specifically, the i th secret in \vec{a} is shared via the t -degree polynomial $A(-i, \mathbf{y})$.

Next, the dealer has to prove, using a distributed zero-knowledge protocol, that indeed $c_i = a_i b_i$ for every i (i.e., that $C(-\beta, 0) = A(-\beta, 0) \cdot B(-\beta, 0)$ for every $\beta \in \{0, \dots, t/2\}$). The input of each party P_j is a point on the univariate polynomials $A(-\beta, \mathbf{y}), B(-\beta, \mathbf{y})$ and $C(-\beta, \mathbf{y})$. The zero-knowledge proof shares and operates on the coefficients of the polynomials used for sharing $\vec{a}, \vec{b}, \vec{c}$. If the dealer shared $\mathcal{O}(M)$ triplets, then the zero-knowledge involves sharing of $\mathcal{O}(Mn)$ values. For simplicity, assume hereafter that $M = \mathcal{O}(n^2)$.

Verifying product relation. We now provide a detailed disposition of the zero-knowledge for product relations via the asynchronous weak-binding secret sharing. For simplicity of notation, the dealer has already (verifiably) shared, for every $u \in U = \{0, \dots, t/2\}^2$, degree- t polynomial $A^u(\mathbf{x}), B^u(\mathbf{x}), C^u(\mathbf{x})$. Each party P_j holds shares $A^u(j), B^u(j), C^u(j)$. The goal of the dealer is to prove that $A^u(0) \cdot B^u(0) = C^u(0)$. The zero-knowledge proof shares and operates on the coefficients of the polynomials used for sharing. Since $|U| \in \mathcal{O}(n^2)$ and each product polynomial has $\mathcal{O}(n)$ coefficients, we have a total of $\mathcal{O}(n^3)$ coefficients to pack. We now need $\mathcal{O}(1)$ trivariate polynomials to pack all needed coefficients. This sharing does not need to produce t -sharing of the secrets. Rather the mere confirmation that the dealer commits to a unique set of secrets is good enough. Hence, as discussed in the previous section,

we can utilize the asynchronous weak-binding secret sharing with light-enhanced features.

Constructing the trivariate polynomials. For every $u \in U$, define:

$$E^u(\mathbf{x}) := A^u(\mathbf{x}) \cdot B^u(\mathbf{x}) - C^u(\mathbf{x}) .$$

We redefine $U = V \times V$ where $V = \{0, \dots, t/2\}$. Then, we need to verify that for every $(\beta, \gamma) \in V \times V$ it holds that

$$E^{(\beta, \gamma)}(0) = A^{(\beta, \gamma)}(0) \cdot B^{(\beta, \gamma)}(0) - C^{(\beta, \gamma)}(0) = 0 ,$$

which implies that $A^{(\beta, \gamma)}(0) \cdot B^{(\beta, \gamma)}(0) = C^{(\beta, \gamma)}(0)$. Since $A^{(\beta, \gamma)}(\mathbf{x})$, $B^{(\beta, \gamma)}(\mathbf{x})$, and $C^{(\beta, \gamma)}(\mathbf{x})$ are polynomials of degree- t , the polynomial $E^{(\beta, \gamma)}(\mathbf{x})$ is of degree- $2t$. We explicitly write

$$E^{(\beta, \gamma)}(\mathbf{x}) = e_1^{(\beta, \gamma)} \mathbf{x} + \dots + e_{2t}^{(\beta, \gamma)} \mathbf{x}^{2t} .$$

Since the constant term of this polynomial is supposed to be 0 (if indeed $A^{(\beta, \gamma)}(0) \cdot B^{(\beta, \gamma)}(0) = C^{(\beta, \gamma)}(0)$) we do not specify it. The dealer embeds the $2t$ coefficients of each of those $(t/2+1)^2$ polynomials in four trivariate polynomials, each of degree $t + t/2$ in \mathbf{x}, \mathbf{y} and \mathbf{z} . Note that each trivariate polynomial can pack $(t/2 + 1)^3$ values, where we have a total of $(t/2 + 1)^2 \cdot 2t$ values to pack; We therefore need four trivariate polynomial $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4$ (in each we pack $t/2(t/2 + 1)^2$ while we can pack $(t/2 + 1)^3$ values; i.e., we do not fully pack it). Specifically (where in all of the following rows we quantify over all $k \in [1, \dots, t/2]$, and $(\beta, \gamma) \in V \times V$):

The coefficients	Embedded in the trivariate	The embedding
$e_1^{(\beta, \gamma)}, \dots, e_{t/2}^{(\beta, \gamma)}$	$\mathbf{S}_1(\mathbf{x}, \mathbf{y}, \mathbf{z})$	$\mathbf{S}_1(-\beta, -k, -\gamma) = e_k^{(\beta, \gamma)}$
$e_{t/2+1}^{(\beta, \gamma)}, \dots, e_t^{(\beta, \gamma)}$	$\mathbf{S}_2(\mathbf{x}, \mathbf{y}, \mathbf{z})$	$\mathbf{S}_2(-\beta, -k, -\gamma) = e_{t/2+k}^{(\beta, \gamma)}$
$e_{t+1}^{(\beta, \gamma)}, \dots, e_{t+t/2}^{(\beta, \gamma)}$	$\mathbf{S}_3(\mathbf{x}, \mathbf{y}, \mathbf{z})$	$\mathbf{S}_3(-\beta, -k, -\gamma) = e_{t+k}^{(\beta, \gamma)}$
$e_{t+t/2+1}^{(\beta, \gamma)}, \dots, e_{2t}^{(\beta, \gamma)}$	$\mathbf{S}_4(\mathbf{x}, \mathbf{y}, \mathbf{z})$	$\mathbf{S}_4(-\beta, -k, -\gamma) = e_{t+t/2+k}^{(\beta, \gamma)}$

This fixes $t/2(t/2 + 1)^2$ points on each trivariate polynomial. The dealer chooses random trivariate polynomials under the above constraints. It then “secret shares” those trivariate polynomials among the parties. Each party P_i receives the share:

$$\text{share}_i = (\mathbf{S}_r(\mathbf{x}, \mathbf{y}, i), \mathbf{S}_r(\mathbf{x}, i, \mathbf{y}), \mathbf{S}_r(i, \mathbf{y}, \mathbf{z}))_{r \in [4]} .$$

Let us assume for simplicity that share_i s are distributed such that *all* the honest parties’ share_i s uniquely define four trivariate polynomials of degree at most $t + t/2$ in each variable. We

have to perform the following checks:

1. The shares that the dealer distributes uniquely define four trivariate polynomials of degree at most $t + t/2$ in each variable.
2. The trivariate polynomials define coefficients of polynomials $E^{(\beta, \gamma)}(\mathbf{x})$ for every $(\beta, \gamma) \in V \times V$. It should hold that for at least $2t + 1$ indices i :

$$a_i^{(\beta, \gamma)} \cdot b_i^{(\beta, \gamma)} - c_i^{(\beta, \gamma)} = E^{(\beta, \gamma)}(i) = i \cdot e_1^{(\beta, \gamma)} + i^2 \cdot e_2^{(\beta, \gamma)} + \dots + i^{2t} \cdot e_{2t}^{(\beta, \gamma)} \quad (5.1)$$

Since each $A^{(\beta, \gamma)}(\mathbf{x})$, $B^{(\beta, \gamma)}(\mathbf{x})$, $C^{(\beta, \gamma)}(\mathbf{x})$ is of degree- t , the polynomial $A^{(\beta, \gamma)}(\mathbf{x}) \cdot B^{(\beta, \gamma)}(\mathbf{x}) - C^{(\beta, \gamma)}(\mathbf{x})$ is of degree- $2t$. If Eq. (5.1) holds for at least $2t + 1$ indices i , then this polynomial is exactly $E^{(\beta, \gamma)}(\mathbf{x})$. Since $E^{(\beta, \gamma)}(0) = 0$, we have that $A^{(\beta, \gamma)}(0) \cdot B^{(\beta, \gamma)}(0) = C^{(\beta, \gamma)}(0)$. To require that the above verification holds for at least $2t + 1$ honest parties, we actually require that it holds for at least $3t + 1$ parties.

To check that the trivariate shares pack the correct values, each P_i must be able to reconstruct $E^{(\beta, \gamma)}(i)$, where

$$\begin{aligned} E^{(\beta, \gamma)}(i) = & \sum_{k=1}^{t/2} \left(i^k \cdot \underbrace{\mathbf{S}_1(-\beta, -k, -\gamma)}_{e_k^{(\beta, \gamma)}} + i^{t/2+k} \cdot \underbrace{\mathbf{S}_2(-\beta, -k, -\gamma)}_{e_{t/2+k}^{(\beta, \gamma)}} \right. \\ & \left. + i^{t+k} \cdot \underbrace{\mathbf{S}_3(-\beta, -k, -\gamma)}_{e_{t+k}^{(\beta, \gamma)}} + i^{t+t/2+k} \cdot \underbrace{\mathbf{S}_4(-\beta, -k, -\gamma)}_{e_{t+t/2+k}^{(\beta, \gamma)}} \right) \end{aligned}$$

We therefore let P_i get the bivariate polynomial from the dealer in addition to share $_i$, which embeds $(E^{(\beta, \gamma)}(i))_{(\beta, \gamma) \in V \times V}$:

$$T_i(\mathbf{x}, \mathbf{z}) = \sum_{r=1}^4 \sum_{k=1}^{t/2} i^{(r-1) \cdot (t/2) + k} \cdot \mathbf{S}_r(\mathbf{x}, -k, \mathbf{z}) . \quad (5.2)$$

Note that for every (β, γ) , $T_i(-\beta, -\gamma) = E^{(\beta, \gamma)}(i)$.

We now need a mechanism for P_i to verify that the dealer indeed passes on a correct bivariate polynomial T_i consistent with the unique trivariate polynomials $(\mathbf{S}_r)_{r \in \{1, 2, 3, 4\}}$ defined by share $_j$ s of the honest parties. We observe that $T_i(\mathbf{x}, j)$ can be computed by P_j based on share $_j$. Since the degree of T_i is $t + t/2$ in both variables, it is enough if $t + t/2 + 1$ honest parties or alternatively a total of $2t + t/2 + 1$ parties confirm that their $T_i(\mathbf{x}, j)$ is consistent with P_i 's received T_i . So, to let P_i verify $T_i(\mathbf{x}, \mathbf{z})$, the parties jointly perform the following:

1. On holding $(\mathbf{S}_r(\mathbf{x}, \mathbf{y}, j))_{r \in [4]}$ as part of share $_j$, for every $r \in [4]$, P_j evaluates $\mathbf{S}_r(\mathbf{x}, \mathbf{y}, j)$

- on $t/2$ values $\mathbf{y} = -1, \dots, -t/2$ to obtain $\mathbf{S}_r(\mathbf{x}, -1, j), \dots, \mathbf{S}_r(\mathbf{x}, -t/2, j)$.
2. Define $t_i^j(\mathbf{x}) = \sum_{r=1}^4 \sum_{k=1}^{t/2} i^{(r-1) \cdot (t/2) + k} \cdot \mathbf{S}_r(\mathbf{x}, -k, j)$.
 3. Send P_i the univariate polynomial $t_i^j(\mathbf{x})$.

P_i can then verify if $t_i^j(\mathbf{x}) = T_i(\mathbf{x}, j)$ for at least $2t + t/2 + 1$ P_j s.

We require the dealer to find a set of size at least $3t + 1$ such that: (a) the honest parties in it must define four unique trivariate polynomials; (b) every honest party in it holds T_i that is consistent with every honest party in the set; and (c) every honest party in it must have successfully verified Equation (5.1). We note that an honest dealer will always be able to find eventually such a set.

Modeling. We describe our protocol in the Simple UC (SUC) framework due to Canetti, Cohen, and Lindell [45]. This implies standard UC security [42]. We try to avoid over-formalism in the protocol descriptions (e.g., we ignore sid while those are implicit). As standard secret-shared protocols in the perfect setting, we conjecture that our protocols are also adaptively secure.

Organization. The rest of the chapter is organized as follows. In Section 5.3, we provide the preliminaries, mainly overview the SUC framework. In Section 5.5, we provide full details of our zero-knowledge protocol, i.e., verifying product relation. In Section 5.6, we provide our rate-1, asynchronous weak-binding secret sharing. We remark again that we do not use this secret sharing directly, and we provide it just for completeness. In Section 5.7, we show verifiable triple sharing, followed by the MPC protocol in Section 5.8.

5.3 Preliminaries

Notations. Our protocols are defined over a finite field \mathbb{F} where $|\mathbb{F}| > n + t + 1$. We denote the elements by $\{-t, \dots, 0, 1, \dots, n\}$. We use $\langle v \rangle$ to denote the degree- t Shamir-sharing of a value v among parties in \mathcal{P} , and $\langle v \rangle_i$ to denote the share held by a party P_i .

5.3.1 Network Model and Notations

We consider an asynchronous network where the parties are $\mathcal{P} = \{P_1, \dots, P_n\}$. The parties are connected via pairwise ideal private channels. To model asynchrony, messages sent on a channel can be arbitrarily delayed, however, they are guaranteed to be eventually received after some finite number of activations of the adversary. In general, the order in which messages are received might be different from the order in which they were sent. Yet, to simplify notation and improve readability, we assume that the messages that a party receives from a channel are guaranteed to be delivered in the order they were sent. This can be achieved

using standard techniques – counters, and acknowledgements, and so we just make this simplification assumption. The detailed description of the modelling and security definition is discussed in Section 2.

Our protocols are defined over a finite field \mathbb{F} where $|\mathbb{F}| > n + t + 1$. We denote the elements by $\{-t, \dots, 0, 1, \dots, n\}$. We use $\langle v \rangle$ to denote the degree- t Shamir-sharing of a value v among parties in \mathcal{P} , and $\langle v \rangle_i$ to denote the share held by a party P_i .

5.3.2 Asynchronous Broadcast and Agreement on a Core Set

A-Cast. Bracha’s asynchronous broadcast (A-Cast) protocol [37] allows a sender to send a message m identically to all the parties at a cost of $\mathcal{O}(n^2\ell)$ where ℓ is the length of the message in bits. If the sender is honest, then all the honest parties eventually terminate with output m . If the sender is corrupt, and some honest party terminates with the output m' , then all the honest parties eventually terminate with the same output m' . In our protocol, the cost of A-Cast can be amortized over multiple instances of triple generation with the same dealer. For ease of description, we use the following representation in our protocols.

- “ P_i broadcasts a message m ” represents an instance of A-Cast with P_i as the sender.
- “ P_j receives a message m from the broadcast of P_i ” represents that P_j outputs m in the instance of A-Cast with P_i as the sender.

Agreement on a Core Set (ACS). The ACS primitive [40] allows parties to agree on a common set of at least $n-t$ parties $\text{Core} \subset \mathcal{P}$, such that each party in Core satisfies some predefined property prop which has the following features:

1. Every honest party eventually satisfies prop .
2. If some honest P_i sees that a party P_j satisfies prop , then eventually all the honest parties see that P_j satisfies prop .

We model ACS as a functionality below. In the functionality, each party i receives (record, i, k) commands with $k \in [n]$ and sends them to the functionality. Each party is guaranteed to receive at least $n-t$ such commands and if some honest i receives a (record, i, k) command, every honest j is guaranteed to eventually receive a (record, j, k) command as well. In addition, parties can receive $\text{receiveS}()$ commands which they forward to the functionality. The functionality then returns a set $S \subseteq [n]$ as a response. Note that in either case, all parties eventually receive the same set of indices $S \subseteq [n]$, such that for every $k \in S$ at least one honest party received a (record, i, k) command.

Functionality 5.3.1: Agreement on a Core Set – \mathcal{F}_{ACS}

The functionality is parameterized by a set of corrupted parties $I \subseteq [n]$. Initialize sets $S_i \leftarrow \emptyset$ for every $i \in [n]$ and $S \leftarrow \perp$. In addition, initialize $\text{returned} \leftarrow 0$.

1. (record, i, k) : Upon receiving this command from party i , add the index k to S_i . Forwards (record, i, k) to the adversary. If $|S_i| \geq n - t$ then set i as ready. If $n - t$ honest parties are ready, then set S to be the set of all indices $k \in [n]$ such that there exists some $\ell \notin I$ for which (record, ℓ, k) was sent.
 2. (set, S') : Upon receiving this command from the adversary, check that $S' \subseteq [n]$ and that $|S'| \geq n - t$. Moreover, check that for every $k \in S'$, there exists some $\ell \notin I$ for which $k \in S_\ell$ (i.e., P_ℓ has submitted (record, ℓ, k)). If all those conditions hold, and $\text{returned} = 0$, then store $S \leftarrow S'$.
 3. $\text{receiveS}()$: Upon receiving this command from some party i , if $S \neq \perp$, set $\text{returned} \leftarrow 1$. Return S .
-

Looking ahead, we use the ACS primitive to identify a common set of parties with the property that a verifiable triple sharing instance and a asynchronous VSS instance initiated by a party in this set must terminate eventually for all the honest parties.

5.3.3 Finding a STAR in a Graph

Definition 5.3.2. Let G be a graph over the nodes $[n]$. We say that a pair $(C, D) \subseteq [n]^2$ such that $C \subseteq D$ is an (n, t) -STAR in G if the following holds:

- $|C| \geq n - 2t$,
- $|D| \geq n - t$,
- For every $c \in C$ and every $d \in D$ the edge (c, d) exists in G .

Canetti [39] showed that if a graph has a clique of size $n - t$, then there exists an efficient algorithm which always finds an (n, t) -star. For completeness, we describe the algorithm below.

Algorithm 5.3.3: STAR Algorithm (G, n, t)

- **Input:** undirected graphs G (over the nodes $\{1, \dots, n\}$), a parameter t .
 1. Find a maximum matching M in \overline{G} (i.e., in the complement graph). Let N be the set of matched nodes (namely, the endpoints of the edges in M).

2. Let T be the set of triangle-heads, i.e., all vertices that are not endpoints of the matching in G , but they have two neighbors in the matching.

$$T := \{i \notin N \mid \exists j, \ell \text{ s.t. } (j, \ell) \in M \text{ and } (i, j), (i, \ell) \in \overline{G}\}.$$

Let $C := [n] \setminus (N \cup T)$.

3. Let D the set of unmatched nodes that have no neighbors in C in \overline{G} . That is, the set:

$$D := \{j \notin N \mid \forall i \in C, (i, j) \notin \overline{G}\}.$$

4. **Output:** If $|C| \geq n - 2t$, and $|D| \geq n - t$ then output (C, D) . Otherwise, output \perp .

Claim 5.3.4. *Let I be a set of cardinality of size at most t . Let G be an undirected graph over $[n]$ such that for every $j, k \notin I$ it holds that $(j, k) \in G$. Then, $C \setminus I$ contains at least $n - 2t$ indices.*

Proof. As in [7] we get that the number of parties in $(N \cup T) \setminus I$ is at most t . Since C is defined as $[n] \setminus (N \cup T)$, we get that $C \setminus I$ is at least $n - 2t$. \square

Claim 5.3.5 ([40]). *Let G be a graph over n vertices that contains a clique of size $n - t$. Then, the algorithm outputs sets (C, D) .*

5.3.4 Bivariate Polynomials

We consider bivariate polynomials of degree $t + q$ in \mathbf{x} and degree t in \mathbf{y} . Such polynomials can be written as follows: $S(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^{t+q} \sum_{j=0}^{t+q} a_{i,j} \mathbf{x}^i \mathbf{y}^j$. Looking ahead, in Section 5.4 we use $q = t$. We have:

Claim 5.3.6. *Let t be a nonnegative integer, let $H \subset [n]$ be a set of cardinality $t + 1$ and let $(f_h(\mathbf{x}))_{h \in H}$ be $t + 1$ univariate polynomials of degree at most $t + q$. Then, there exists a unique bivariate polynomial $S(\mathbf{x}, \mathbf{y})$ of degree $t + q$ in \mathbf{x} and degree t in \mathbf{y} satisfying for every $h \in H$: $S(\mathbf{x}, h) = f_h(\mathbf{x})$.*

5.3.5 Trivariate Polynomials

We consider trivariate polynomial of degree $t + q$ in variables $\mathbf{x}, \mathbf{y}, \mathbf{z}$. Such polynomial can be written as follows:

$$S(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{i=0}^{t+q} \sum_{j=0}^{t+q} \sum_{k=0}^{t+q} a_{i,j,k} \mathbf{x}^i \mathbf{y}^j \mathbf{z}^k.$$

Looking ahead, in our protocol we consider $q = t/2$.

Claim 5.3.7 (Interpolation). *Let t be a nonnegative integer, let $H \subset [n]$ be a set of cardinality $t + q + 1$, and let $(S_h(\mathbf{x}, \mathbf{y}))_{h \in H}$ be $t + q + 1$ bivariate polynomials of degree at most $t + q$ (in both \mathbf{x}, \mathbf{y}) each. Then, there exists a unique trivariate polynomial $S(\mathbf{x}, \mathbf{y}, \mathbf{z})$ of degree $t + q$ such that for every $h \in H$:*

$$S(\mathbf{x}, \mathbf{y}, h) = S_h(\mathbf{x}, \mathbf{y})$$

Proof. Define the trivariate polynomial $S(\mathbf{x}, \mathbf{y}, \mathbf{z})$ via a generalization of the Lagrange interpolation. For every $h \in H$, define the trivariate polynomial $S_h(\mathbf{x}, \mathbf{y}, \mathbf{z})$ as follows:

$$S_h(\mathbf{x}, \mathbf{y}, \mathbf{z}) = S_h(\mathbf{x}, \mathbf{y}) \cdot \frac{\prod_{j \in H \setminus \{h\}} (\mathbf{z} - j)}{\prod_{j \in H \setminus \{h\}} (h - j)}.$$

Note that $S_h(\mathbf{x}, \mathbf{y}, h) = S_h(\mathbf{x}, \mathbf{y})$, and for every $j \in H \setminus \{h\}$, $S_h(\mathbf{x}, \mathbf{y}, j) = 0$. Moreover, $S_h(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is of a trivariate polynomial of degree $t + q$. Then, define:

$$S(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{h \in H} S_h(\mathbf{x}, \mathbf{y}, h).$$

Clearly, for every $h \in H$ it holds that $S(\mathbf{x}, \mathbf{y}, h) = S_h(\mathbf{x}, \mathbf{y})$, and $S(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is a trivariate polynomial of degree $t + q$. We now show that S is unique. Assume that there exist two different polynomials $S_1(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and $S_2(\mathbf{x}, \mathbf{y}, \mathbf{z})$ that satisfy the conditions in the claim. Consider the polynomial $R(\mathbf{x}, \mathbf{y}, \mathbf{z}) = S_1(\mathbf{x}, \mathbf{y}, \mathbf{z}) - S_2(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Since S_1 and S_2 are two trivariate polynomial of degree $t + q$ then R is also a trivariate polynomial of degree $t + q$. Moreover, for every $h \in H$ it holds that $R(\mathbf{x}, \mathbf{y}, h) = 0$. For every $\alpha, \beta \in \mathbb{F}$, consider the degree $t + q$ univariate polynomial $R(\alpha, \beta, \mathbf{z})$. It holds that $R(\alpha, \beta, h) = 0$ for all $h \in H$, and thus it equals 0 on $t + q + 1$ points, i.e., this is the all-zero univariate polynomial. Moreover, for every $\alpha, \beta, \gamma \in \mathbb{F}$ it holds that $R(\alpha, \beta, \gamma) = 0$, that is, $R(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is the all-zero polynomial and thus $S_1(\mathbf{x}, \mathbf{y}, \mathbf{z}) = S_2(\mathbf{x}, \mathbf{y}, \mathbf{z})$. \square

5.4 Verifiable Packed Bivariate Secret Sharing

In this section, we provide a protocol for packing $\mathcal{O}(n)$ secretes in a bivariate polynomial. We model the packed VSS in Functionality 5.4.1 below. The protocol is a simplification of the protocol of [50]: instead of the dealer re-broadcasting sets, we show how to do it with just a single broadcast message. Moreover, as opposed to [50], we provide full simulation in the SUC framework.

Functionality 5.4.1: Packed VSS Functionality – \mathcal{F}_{VSS}

The functionality is parameterized by the set of corrupted parties $I \subseteq [n]$.

1. The dealer sends to the functionality its input $S(\mathbf{x}, \mathbf{y})$.
 2. The functionality verifies that S is of degree at most $3t/2$ in \mathbf{x} and degree at most t in \mathbf{y} . If not, it does not terminate.
 3. If the above condition does hold, then the functionality sends to the ideal adversary the shares $S(\mathbf{x}, i), S(i, \mathbf{y})$ for every $i \in I$, and for each honest party P_j it sends $S(\mathbf{x}, j), S(j, \mathbf{y})$.
-

Definition 5.4.2. For an undirected graph $G = (V, E)$ with $V = [n]$, we say that the four sets (C, D, G, F) are valid if the following conditions are satisfied:

1. $|C| \geq 2t + 1$, $C \subseteq D$, and $|D|, |G|, |F| \geq 3t + 1$.
 2. For every $c \in C$ and $d \in D$ it holds that (c, d) is an edge in G .
 3. For every $g \in G$ it holds that $|\Gamma(g) \cap C| \geq 2t + 1$.
 4. For every $f \in F$ it holds that $|\Gamma(f) \cap G| \geq 3t + 1$.
-

Protocol 5.4.3: Packed VSS Protocol – Π_{pVSS}

- **Input:** The dealer holds a bivariate polynomial $S(\mathbf{x}, \mathbf{y})$ of degree $3t/2$ in \mathbf{x} and degree t in \mathbf{y} . Other parties hold no input.
- **The protocol:**
 1. **(Dealing Shares):** The dealer sends (shares, $D, i, (f_i(\mathbf{x}), g_i(\mathbf{y}))$) with $f_i(\mathbf{x}) = S(\mathbf{x}, i)$ and $g_i(\mathbf{y}) = S(i, \mathbf{y})$ to each party P_i .
 2. **(Pairwise Consistency Checks:)** Each party P_i does as follows
 - (a) Upon receiving shares from the dealer, P_i verifies that $f_i(\mathbf{x})$ is of degree at most $3t/2$ and $g_i(\mathbf{y})$ is of degree at most t . If so, P_i sends (exchange, $i, j, f_i(j), g_i(j)$) to every P_j .
 - (b) Upon receiving shares (exchange, $j, i, u_{j,i}, v_{j,i}$) from P_j , check that $u_{j,i} = g_i(j)$ and $v_{j,i} = f_i(j)$. If so, broadcast the message $\text{Good}(i, j)$.
 3. **(Valid (C, D, G, F) finding:)** The dealer does as follows
 - (a) Initialize an undirected graph G where $V = [n]$. Upon viewing broadcasted messages $\text{Good}(k, \ell)$ broadcasted by P_k and $\text{Good}(\ell, k)$ broadcasted by P_ℓ , add the edge (k, ℓ) to G .
 - (b) Run the STAR algorithm (Algorithm 5.3.3) to find sets $(C, D) \subset [n]^2$ where $|C| \geq 2t + 1$, $|D| \geq 3t + 1$, and for every $c \in C$ and $d \in D$, the edge (c, d) is in G .

- (c) Let G be the set of parties that agree with at least $2t + 1$ parties in C . That is, initialize $G = \emptyset$ and add i to G if $|\Gamma(i) \cap C| \geq 2t + 1$.
 - (d) Let F be the set of parties that agree with at least $3t + 1$ parties in G . That is, initialize $F = \emptyset$ and add i to F if $|\Gamma(i) \cap G| \geq 3t + 1$.
 - (e) If $|C| \geq 2t + 1$, $C \subseteq D$, and $|D|, |G|, |F| \geq 3t + 1$, then broadcast (C, D, G, F) . Otherwise, continue to listen to Good messages, and repeat.
4. **(Verifying (C, D, G, F) .)** Each party P_i :
- (a) Initialize an undirected graph G_i where $V = [n]$. Upon viewing broadcasted messages $\text{Good}(k, \ell)$ from P_k and $\text{Good}(\ell, k)$ from P_ℓ , add the edge (k, ℓ) to G_i .
 - (b) Check if (C, D, G, F) is valid for the graph G_i . If not, then continue to listen to Good message, and with each new edge, re-check validation.
5. **(Deciding on output)** Once (C, D, G, F) are valid for G_i , P_i outputs its share as follows and terminates subsequently
- **(Parties $i \in G \cap F$)** Output $(f_i(\mathbf{x}), g_i(\mathbf{y}))$.
 - **(Parties $i \notin G$)**
 - (a) Consider the messages $(u_{j,i}, v_{j,i})$ from Step 2b for parties $j \in F$, and using Reed Solomon decoding, try to decode the unique univariate polynomial $g_i(\mathbf{y})$ of degree- t satisfying $g_i(j) = u_{j,i}$ for all but t values in F . If there is no unique decoding, wait to receive additional points $(u_{j,i}, v_{j,i})$.
 - (b) Once decoded, send $(\text{reconstruct}, i, k, g_i(k))$ to each party P_k for $k \notin F$.
 - **(Parties $i \notin F$)**:
 - (a) Consider all points $(j, v_{j,i})$ obtained from messages $(\cdot, v_{j,i})$ from Step 2b for parties $j \in G$, or points $(j, v_{j,i})$ obtained from messages $(\text{reconstruct}, j, i, v_{j,i})$ from parties not in G . Use Reed Solomon decoding procedure to decode the unique univariate polynomial $f_i(\mathbf{x})$ of degree $3t/2$ satisfying $f_i(j) = v_{j,i}$ for all but t parties. If there is no unique decoding, wait to receive additional points $(u_{j,i}, v_{j,i})$ or $(\text{reconstruct}, \ell, i, v_{\ell,i})$.
 - Once both $f_i(\mathbf{x})$ (for $i \notin F$) and $g_i(\mathbf{y})$ (for $i \notin G$) were reconstructed, then terminate and output $(f_i(\mathbf{x}), g_i(\mathbf{y}))$.
-

Theorem 5.4.4. *Let $n \geq 4t + 1$. Protocol 5.4.3, Π_{pVSS} , securely computes \mathcal{F}_{VSS} (Functionality 5.4.1) in the presence of a malicious adversary controlling at most t parties. It has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits*

of broadcast for sharing $\mathcal{O}(n)$ values simultaneously. Each party broadcasts at most $\mathcal{O}(n \log n)$ bits.

Before showing full simulatability, we have the following two claims regarding the protocol: validity and binding. We will use those claims in the proof of theorem 5.4.4 when we show full simulation.

Claim 5.4.5 (Validity). *If the dealer is honest, then the protocol eventually terminates, and each honest party P_i output $S(\mathbf{x}, i), S(i, \mathbf{y})$, where $S(\mathbf{x}, \mathbf{y})$ is the input of the dealer.*

Proof. Eventually, each honest P_j broadcast $\text{Good}(j, \ell)$ for all honest parties $\ell \notin I$. We show that the dealer must eventually broadcast (C, D, G, F) . Consider time T in which all Good messages between pairs of honest parties were received by the dealer. There are two cases to consider:

Case I – The dealer broadcast (C, D, G, F) before time T . In that case, the dealer terminates before time T . Claim 5.4.6 below shows that all honest parties output shares that lie on the same bivariate polynomial. Moreover, this polynomial is determined by the set of honest parties in C , which is the bivariate polynomial $S(\mathbf{x}, \mathbf{y})$ that the dealer holds.

Case II – The dealer did not broadcast (C, D, G, F) before time T . At time T , the dealer is guaranteed to find an (n, t) -star (C, D) such that $|C| \geq n - 2t \geq 2t + 1$, and $|D| \geq n - t \geq 3t + 1$ (see Claim 5.3.4). Moreover, C contains at least $2t + 1$ honest parties. Since G is defined as the set of parties that agree with at least $2t + 1$ in C , we get that the set G must contain all honest parties. Moreover, the set F is defined as the set of parties that agree with at least $3t + 1$ parties in G . Since G contains all honest parties (i.e., at least $3t + 1$ parties), all honest parties are part of F . We conclude that the dealer finds and eventually broadcasts (C, D, G, F) .

All honest parties will eventually receive the shares from the dealer, and each pair of honest parties P_k, P_ℓ will eventually broadcast the messages $\text{Good}(k, \ell)$ and $\text{Good}(\ell, k)$. Therefore, the graph of the dealer G , eventually contains a clique of all honest parties (i.e., of $3t + 1$). Once this occurs, each honest party is both in G and F , and therefore output the same shares as received from the dealer. Since the dealer chose all polynomials to lie on the same bivariate polynomial $S(\mathbf{x}, \mathbf{y})$, we have that $(f_i(\mathbf{x}), g_i(\mathbf{y})) = (S(\mathbf{x}, i), S(i, \mathbf{y}))$. \square

Claim 5.4.6 (Binding). *When one honest party terminates, all honest parties will eventually terminate with shares on the same bivariate polynomial $S(\mathbf{x}, \mathbf{y})$. The bivariate polynomial $S(\mathbf{x}, \mathbf{y})$ is interpolated from the shares that the lexicographically first $t + 1$ honest parties in C received from the dealer, where C is defined as the set that the dealer broadcasts.*

Proof. Consider the first honest party that terminates, say some P_{j^*} . If P_{j^*} terminates, then the dealer must have broadcasted sets (C, D, G, F) , and P_{j^*} must have validated the property in its local graph.

Let $H \subseteq C$ be a set of some $t + 1$ honest parties in C . Since $|C| \geq 2t + 1$, there must exist such a set H . Consider the unique bivariate polynomial $S(\mathbf{x}, \mathbf{y})$ satisfying $S(\mathbf{x}, h) = f_h(\mathbf{x})$ for every $h \in H$, see Claim 5.3.6. At this point:

1. **For every honest $d \in D$ it holds that $g_d(\mathbf{y}) = S(d, \mathbf{y})$.** Specifically, $g_d(\mathbf{y}), S(d, \mathbf{y})$ are two univariate polynomials of degree- t , and for every $h \in H$ it holds that $g_d(h) = f_h(d) = S(d, h)$, as otherwise the edge (d, h) does not exist in G . Since the two polynomials agree on $t + 1$ points, they are the same polynomial.
2. **For every honest $c \in C$ it holds that $f_c(\mathbf{x}) = S(\mathbf{x}, c)$.** This trivially holds for the parties in $H \subseteq C$. We show that this also holds for the other honest parties in C . Specifically, for every honest $c \in C$ and $d \in D$, P_c and P_d checked that $f_c(d) = g_d(c)$. Therefore, the two degree- $3t/2$ univariate polynomials, $f_c(\mathbf{x})$ and $S(\mathbf{x}, c)$ agree on $2t + 1$ points, and therefore it holds that $f_c(\mathbf{x}) = S(\mathbf{x}, c)$.
3. **For every honest $j \in G$ it holds that $g_j(\mathbf{y}) = S(\mathbf{y}, j)$.** Each party in G agrees with at least $2t + 1$ parties in C . Therefore, P_j agrees with at least $t + 1$ honest parties in C . Thus, the two polynomials $g_j(\mathbf{y})$ and $S(\mathbf{y}, j)$ must agree on $t + 1$ points (i.e., each such honest $c \in C$ that agrees with P_j exchanged $g_j(c)$ and $f_c(j)$ and it holds that $g_j(c) = f_c(j) = S(c, j)$). As a result, $g_j(\mathbf{y}) = S(\mathbf{y}, j)$.
4. **For every honest $j \in F$ it holds that $f_j(\mathbf{x}) = S(\mathbf{x}, j)$.** Each party in F agrees with at least $3t + 1$ parties in G . Since $|G| \geq 3t + 1$, we get that P_j agrees with at least $2t + 1$ honest parties in G . For each such P_j and an honest $k \in G$ it holds that $f_j(k) = g_k(j) = S(j, k)$. We get that two degree- $3t/2$ polynomials $f_j(\mathbf{x})$ and $S(\mathbf{x}, j)$ must agree on $2t + 1$ points, and therefore $f_j(\mathbf{x}) = S(\mathbf{x}, j)$.

Since all Good messages are broadcasted, each honest party will eventually see the same edges as the first honest party that terminated, P_{j^*} , and the set (C, D, G, F) will also be validated by each one of the honest parties.

We get that all honest parties $j \in F$ hold a polynomial $f_j(\mathbf{x}) = S(\mathbf{x}, j)$. Moreover, all honest parties $j \in G$ holds a polynomial $g_j(\mathbf{y}) = S(\mathbf{y}, j)$. If $j \in G \cap F$, then it has both $(f_j(\mathbf{x}), g_j(\mathbf{y}))$ and it can just terminate. For parties that do not hold both polynomials:

1. If $j \notin G$, then it considers all the points that it received from parties in F in Step 2b. Those points lie on the polynomial $S(j, \mathbf{y})$. Since G contains $3t + 1$ parties, it contains at least $2t + 1$ honest parties. Moreover, P_j might receive at most t incorrect points.

Therefore, the decoding algorithm must eventually have a unique decoding, to the polynomial $S(j, \mathbf{y})$. Therefore, P_j eventually obtains $g_j(y) = S(j, \mathbf{y})$. At that point it sends (reconstruct, $j, g_j(k)$) for every $k \notin F$, where $g_j(k) = S(j, k)$.

2. If $j \notin F$, then it considers all the points from parties in G as received from Step 2b, and in addition it considers the points from parties not in G , after they reconstruct their correct g polynomial. As a result, P_j is guaranteed to eventually receive $3t + 1$ correct points on the polynomial $S(\mathbf{x}, j)$, and it would have at most t incorrect points. Reed Solomon decoding is guaranteed to succeed.

We conclude that eventually, all honest parties terminate and output shares $(f_i(\mathbf{x}), g_i(\mathbf{y})) = (S(\mathbf{x}, i), S(i, \mathbf{y}))$. \square

Proof. We separate the analysis to the case of an honest dealer and a corrupted dealer.

Case I – the case of an honest dealer. We present the simulator \mathcal{S} :

1. Upon activation, the simulator invokes the adversary \mathcal{A} .
2. The simulator receives from the functionality the shares $(f_i(\mathbf{x}), g_i(\mathbf{y})) = (S(\mathbf{x}, i), S(i, \mathbf{y}))$ for every $i \in I$. Moreover, it receives requests from the router to deliver the outputs of the functionality to the honest parties. The simulator will allow to deliver the outputs as the protocol proceed.
3. \mathcal{S} sends to the adversary \mathcal{A} the message (shares, $D, i, f_i(\mathbf{x}), g_i(\mathbf{y})$) for every $i \in I$ as coming from the dealer.
4. Send to the adversary \mathcal{A} all delivery requests (shares, D, j) for $j \notin I$ as coming from the router.
5. Once \mathcal{A} delivers (shares, D, j), simulate P_j sending to each corrupted party P_i the message (exchange, $j, i, f_j(i), g_j(i)$) where $(f_j(i), g_j(i)) = (S(j, i), S(i, j)) = (g_i(j), f_i(j))$. Moreover, simulate P_j sending (exchange, j, k) for every $k \notin I$.
6. Once \mathcal{A} delivers the message (exchange, j, k), simulate P_k broadcasting Good(k, j).
7. Once the adversary sends (exchange, $i, j, u_{i,j}, v_{i,j}$), verify that $u_{i,j} = f_i(j)$ and $v_{i,j} = g_i(j)$. If so (and P_j already received its shares from the dealer), simulate P_j broadcasting Good(j, i).
8. Initialize an undirected graph G over nodes $[n]$. For each message Good(j, k) broadcasted (by the adversary or a simulated honest party), if the message Good(k, j) was also delivered to the dealer, then add the edge (j, k) to G . Check whether the graph G contains sets (C, D, G, F) as in the protocol. If so, simulate the dealer broadcasting (C, D, G, F) .

9. For each party P_j , simulate the following:
 - (a) P_j initializes a graph G_i . Once \mathcal{A} delivers the broadcasted messages $\text{Good}(j, k)$ and $\text{Good}(k, j)$, add the edge (k, j) to G_i . Moreover, once the adversary delivers the broadcasted message (C, D, G, F) to P_j , check that (C, D, G, F) is valid in G_i . If not, continue to listen to addition Good messages.
 - (b) Once (C, D, G, F) is valid for P_j , if $j \in G \cap F$, then allow the router to deliver the output of party P_j from the ideal functionality to the dummy party.
 - (c) If $j \notin G$, then wait until the adversary delivers additional $(\text{exchange}, k, j)$ messages for honest $k \notin I$. Moreover, the simulator can identify the number of incorrect shares that the adversary sent to P_j via $(\text{exchange}, i, j, \cdot, \cdot)$. Let $t' \leq t$ be that number of incorrect shares. Once the adversary allows receiving $t' + t + 1$ shares from $F \setminus I$, then P_j can reconstruct its polynomial $g_j(\mathbf{y})$. Simulate P_j sending $(\text{reconstruct}, j, k, g_j(k))$ for each honest party P_k for $k \notin F$ by giving the adversary \mathcal{A} the messages $(\text{reconstruct}, j, k)$ as coming from the router.
 - (d) If $j \notin F$, then continue to listen to all messages $(\text{exchange}, k, i)$ delivered from the adversary, and all $(\text{reconstruct}, \ell, i)$. Moreover, the simulator can identify the number of incorrect sub-shares that the adversary had sent P_j via $(\text{exchange}, i, j)$ messages. Let t' be the number of incorrect shares. Once the adversary delivers to P_j exactly $t' + 3t/2 + 1$ shares from honest parties, then P_j can reconstruct $f_j(\mathbf{x})$.
 - (e) Once P_j can reconstruct (if needed) $f_j(\mathbf{x})$ and $g_j(\mathbf{y})$, the simulator \mathcal{S} allows the router in the ideal model to deliver the output of P_j from the functionality to P_j .
10. The adversary must eventually allow delivering all messages, in then each simulated honest party P_j eventually delivers all messages sent from the functionality to the honest parties in the ideal execution.

We now show that the view of the environment \mathcal{Z} is the same in both executions. From inspection, it is easy to see that the view of the adversary \mathcal{A} is the same in both execution. In particular, this is due to the fact that the protocol and the simulation are deterministic, and that the messages of the honest parties to the corrupted parties can be simulated from the shares of the corrupted parties. Our goal is to show now that the outputs of the parties is the same between the two executions (and in particular, each party receives its output at the same “time” in both executions). Regarding the stage in which each party receives its output, we remark that this is determined by the adversary and how it instructs the router in the real world, or the simulated router in the ideal world (in which the simulator then forwards the request to deliver to the real router in the ideal world).

Therefore, it suffices to show that the outputs of the honest parties are the same in both executions. Claim 5.4.5 shows that in the real world, each honest party P_j eventually receives shares $S(\mathbf{x}, j), S(j, \mathbf{y})$, while the scheduling is determined by the adversary. In the ideal, the trusted party delivers to each honest party P_j the shares $S(\mathbf{x}, j), S(j, \mathbf{y})$. The adversary \mathcal{A} decides on the scheduling (via \mathcal{S} and the router), but since the view of \mathcal{A} is exactly the same in the real and ideal, the scheduling is exactly the same.

Case II - the case of a corrupted dealer. We provide the simulator \mathcal{S} :

1. Upon activation, the simulator activates the adversary \mathcal{A} .
2. Since the protocol is deterministic and the honest parties have no inputs, the simulator can simulate all honest parties (and the router) in an execution with \mathcal{A} . This means that it listens to the messages \mathcal{A} sends to the honest parties, and it can also simulate the messages between honest parties (where \mathcal{A} receives notifications from the simulated router). Note that the simulation might not terminate.
3. When the first honest party P_{j^*} terminates, then all honest parties will eventually terminate (see Claim 5.4.6). Consider the set C that dealer has broadcasted in the simulated execution. Interpolate the polynomial $S(\mathbf{x}, \mathbf{y})$ according to the lexicographically first $t + 1$ honest parties in C , similarly to the binding property in the proof of Claim 5.4.6. Send to the functionality the polynomial $S(\mathbf{x}, \mathbf{y})$. It is guaranteed that S has degree at most $2t$ in \mathbf{x} and degree at most t in \mathbf{y} .
4. The functionality delivers to each honest party P_j . The simulator instructs the router to deliver the output to P_{j^*} .
5. It continues to simulate the protocol to the adversary, where whenever a simulated honest party P_j obtains an output in the simulated protocol, then the simulator \mathcal{S} delivers to the router to send the output to the dummy party P_j in the ideal.
6. Eventually, all honest parties will receive output in the simulated execution. The simulator then terminates.

Clearly, the view of the adversary is exactly the same in the real and ideal executions, as the honest parties have no inputs and are deterministic. If some honest party terminates, then as follows from the proof of Claim 5.4.6, all honest parties eventually output shares that all lie on a bivariate polynomial that is defined from the lexicographically first $t + 1$ honest parties in C . We note that the environment \mathcal{Z} sees that the outputs at the same activation in the real and ideal. Specifically, upon the activation in which the first honest party P_{j^*} terminates, in the ideal execution the simulator extracts the input to send to the trusted party, and it

delivers the output to P_{j^*} in the ideal. Since the protocol proceeds in the same way in both executions, parties receive their outputs at the same activations. \square

5.5 Verifying Product Relation

In this section, we show how to realize the product-relation verification functionality. Assume that a dealer owns and preshares $\mathcal{O}(n^2)$ Shamir-sharings of triples, such that each of the triples are supposed to satisfy a product relation. That is, for a triple (a, b, c) , c must be equal to ab . The parties input shares of those shared triples to the functionality, and the functionality checks that shares define triples satisfying product relations.

Functionality 5.5.1: Verifying Product Relation – $\mathcal{F}_{\text{ProdVer}}$

The functionality is parameterized by a set of corrupted parties $I \subseteq [n]$.

1. Let $U = [(t/2+1)^2]$. Each party P_j sends to the functionality a set of points $(a_j^u, b_j^u, c_j^u)_{u \in U}$.
2. For every $u \in U$, the functionality reconstructs the unique degree- t univariate polynomials $A^u(\mathbf{x}), B^u(\mathbf{x}), C^u(\mathbf{x})$ satisfying

$$A^u(j) = a_j^u, \quad B^u(j) = b_j^u, \quad C^u(j) = c_j^u.$$

If the dealer is honest, then the dealer also sends $A^u(\mathbf{x}), B^u(\mathbf{x}), C^u(\mathbf{x})$.

3. If the dealer is honest, then give the adversary the shares $A^u(i), B^u(i)$ and $C^u(i)$ for every $i \in I$. If the dealer is corrupted, then give the adversary the reconstructed $A^u(\mathbf{x}), B^u(\mathbf{x}), C^u(\mathbf{x})$.
4. The functionality verifies that for every $u \in U$ it holds that

$$A^u(0) \cdot B^u(0) = C^u(0).$$

If yes, then it sends OK to all parties and halts. Otherwise, the functionality never terminates.

5.5.1 Trivariate Polynomial Verification – Functionality

We overviewed the ZK property in Section 5.2.4. Towards realizing Functionality 5.5.1, we introduce an aiding functionality. To recall, we write $U = V \times V$ where $V = \{0, \dots, t/2\}$, and the dealer embeds the coefficients of the polynomials $A^{(\beta, \gamma)}, B^{(\beta, \gamma)}, C^{(\beta, \gamma)}$ in four trivariate

polynomials. We abstract some computations that the parties perform to improve readability by considering general predicates. Specifically:

1. Each party P_i also receives from the dealer the bivariate polynomial $T_i(\mathbf{x}, \mathbf{z})$ as part of its share; each party P_j computes from its share share_j a univariate polynomial that is supposed to be $T_i(\mathbf{x}, j)$ by applying some linear combination over its shares. We abstract this linear combination as “linear circuit”, formally defined below. Note that the same computation that the dealer performs on the trivariate shares to obtain $T_i(\mathbf{x}, \mathbf{y})$, each party performs on its share share_j to obtain $T_i(\mathbf{x}, j)$.
2. Each party P_i also checks that for every (β, γ) it holds that $T_i(-\beta, -\gamma) = a_i^{(\beta, \gamma)} \cdot b_i^{(\beta, \gamma)} - c_i^{(\beta, \gamma)}$. We abstract this check as an “external validity” predicate that P_i enters as input.

Linear circuits. We consider a circuit L_j that receives as an input a bivariate polynomial $F(\mathbf{x}, \mathbf{y})$ and it has the following structure:

1. Evaluate $F(\mathbf{x}, \mathbf{y})$ on several constants $\mathbf{y} = \alpha_1, \dots, \alpha_k$ in the field. The results are univariate polynomials $f_1(\mathbf{x}) = F(\mathbf{x}, \alpha_1), \dots, f_k(\mathbf{x}) = F(\mathbf{x}, \alpha_k)$.
2. Output the univariate polynomial $f^{L_j}(\mathbf{x}) = \sum_{\ell=1}^k \lambda_\ell \cdot f_\ell(\mathbf{x})$ for some constants $\lambda_1, \dots, \lambda_k$. That is, output a fixed linear combination of $f_1(\mathbf{x}), \dots, f_k(\mathbf{x})$.

We write $f^{L_j(F)}(\mathbf{x}) = L_j(F(\mathbf{x}, \mathbf{y}))$. We also evaluate the circuit L_j on a trivariate polynomial $\mathbf{F}(\mathbf{x}, \mathbf{y}, \mathbf{z})$. In that case:

1. Evaluate $\mathbf{F}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ on the same constants $\mathbf{y} = \alpha_1, \dots, \alpha_k$. The results are bivariate polynomials $(F_1(\mathbf{x}, \mathbf{z}), \dots, F_k(\mathbf{x}, \mathbf{z})) = (\mathbf{F}(\mathbf{x}, \alpha_1, \mathbf{z}), \dots, \mathbf{F}(\mathbf{x}, \alpha_k, \mathbf{z}))$.
2. Output the bivariate polynomial which is a fixed linear combination $F^{L_j}(\mathbf{x}, \mathbf{z}) := \sum_{\ell=1}^k \lambda_\ell \cdot F_\ell(\mathbf{x}, \mathbf{z})$.

We write $F^{L_j(\mathbf{F})}(\mathbf{x}, \mathbf{y}) = L_j(\mathbf{F}(\mathbf{x}, \mathbf{y}, \mathbf{z}))$. For every $i \in [n]$, consider $F_i(\mathbf{x}, \mathbf{y}) = \mathbf{F}(\mathbf{x}, \mathbf{y}, i)$, and let $f^{L_j(F_i)}(\mathbf{x}) = L_j(F_i(\mathbf{x}, \mathbf{y}))$. Then clearly it holds that

$$f^{L_j(F_i)}(\mathbf{x}) = L_j(F_i(\mathbf{x}, \mathbf{y})) = L_j(\mathbf{F}(\mathbf{x}, \mathbf{y}, i)) = F^{L_j(\mathbf{F})}(\mathbf{x}, i) .$$

The specific linear circuit that we use is given below in Circuit 5.5.2.

External validity. The predicate $\text{ExternalValidity}_j$ receives as input the share of P_j and outputs 0 or 1. The exact predicate that we use is given in Algorithm 5.5.3.

Linear Circuit 5.5.2 (The circuit L_i):

- **Input:** Trivariate polynomials $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4$.
 1. For $r \in [1, \dots, 4]$, evaluate $\mathbf{S}_r(\mathbf{x}, \mathbf{y}, \mathbf{z})$ on the constants $\mathbf{y} = -1, \dots, -t/2$.
 2. Obtain $\mathbf{S}_r(\mathbf{x}, -k, \mathbf{z})$ for $k \in [1, \dots, t/2]$ and $r \in [1, \dots, 4]$.
 3. Define $T_i(\mathbf{x}, \mathbf{z}) := \sum_{r=1}^4 \sum_{k=1}^{t/2} i^{(r-1) \cdot (t/2) + k} \cdot \mathbf{S}_r(\mathbf{x}, -k, \mathbf{z})$.
- **Output:** The bivariate polynomial $T_i(\mathbf{x}, \mathbf{z})$.

Algorithm 5.5.3: The predicate $\text{ExternalValidity}_i$

- **Input:** The share share_i of P_i , which consists of shares on each one of the polynomials $\mathbf{S}_1, \dots, \mathbf{S}_4$ and bivariate polynomial $T_i(\mathbf{x}, \mathbf{z})$.
- **Parameters:** For every $(\beta, \gamma) \in V \times V$ the algorithm is hardwired with values $a_i^{(\beta, \gamma)}, b_i^{(\beta, \gamma)}, c_i^{(\beta, \gamma)} \in \mathbb{F}$.
- **The algorithm:** Output 1 iff for every $\beta, \gamma \in V$ it holds that:

$$T_i(-\beta, -\gamma) = a_i^{(\beta, \gamma)} \cdot b_i^{(\beta, \gamma)} - c_i^{(\beta, \gamma)} .$$

We are now ready to provide the functionality which the product-relation verification protocol uses as its main building block:

Functionality 5.5.4: Trivariate Polynomial Verification – $\mathcal{F}_{\text{TriVer}}$

The functionality is parameterized by (1) The set of corrupted parties $I \subseteq [n]$; (2) Some linear circuits L_1, \dots, L_n as defined above. The functionality works as follows:

1. The dealer sends four trivariate polynomials $\mathbf{S}_1(\mathbf{x}, \mathbf{y}, \mathbf{z}), \dots, \mathbf{S}_4(\mathbf{x}, \mathbf{y}, \mathbf{z})$.
2. Define the share of each party P_i to be

$$\text{share}_i = \left((\mathbf{S}_r(\mathbf{x}, \mathbf{y}, i), \mathbf{S}_r(\mathbf{x}, i, \mathbf{z}), \mathbf{S}_r(\mathbf{x}, \mathbf{y}, i))_{r \in [4]}, L_i(\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4) \right) .$$

If the dealer is honest, then for every $i \in I$ send P_i the share share_i .

3. The honest parties send to the functionality their external validity predicates $\text{ExternalValidity}_j$ to the functionality. Each $\text{ExternalValidity}_j$ takes as input share_j and outputs 0 or 1.
4. If the dealer is corrupted, then send $(\text{ExternalValidity}_j)_{j \notin I}$ to the adversary.
5. If each one of the four trivariate polynomials $\mathbf{S}_1, \dots, \mathbf{S}_4$ is of degree $t + t/2$ in each one of the three variables, and $\text{ExternalValidity}_j(\text{share}_j) = 1$ holds for at least $2t + 1$ honest

parties, then send OK to all parties and halt. Otherwise, the functionality does not terminate.

5.5.2 Verifying Product Relation using Trivariate Polynomial

We now proceed to show how to implement Functionality 5.5.1, using the trivariate sharing as a building block (i.e., Functionality 5.5.4). We then provide the theorem statement which we prove subsequently.

Protocol 5.5.5: Verifying the Product Relation – Π_{ProdVer}

- **Input:** The dealer holds $A^{(\beta,\gamma)}(\mathbf{x}), B^{(\beta,\gamma)}(\mathbf{x}), C^{(\beta,\gamma)}(\mathbf{x})$ for every $(\beta, \gamma) \in V \times V$. Each party P_i holds the points $A^{(\beta,\gamma)}(i), B^{(\beta,\gamma)}(i), C^{(\beta,\gamma)}(i)$ for every $(\beta, \gamma) \in V \times V$.
 - **The protocol:**
 1. The dealer computes $E^{(\beta,\gamma)}(\mathbf{x}) = A^{(\beta,\gamma)}(\mathbf{x}) \cdot B^{(\beta,\gamma)}(\mathbf{x}) - C^{(\beta,\gamma)}(\mathbf{x})$ for every $(\beta, \gamma) \in V \times V$ and define the coefficients $e_1^{(\beta,\gamma)}, \dots, e_{2t}^{(\beta,\gamma)}$.
 2. The dealer chooses four trivariate polynomials S_1, \dots, S_4 of degree $t + t/2$ in all three variables uniformly at random while embedding the coefficients $e_1^{(\beta,\gamma)}, \dots, e_{2t}^{(\beta,\gamma)}$ as described in the text in Section 5.2.4.
 3. The parties invoke Functionality 5.5.4 where the dealer inputs S_1, \dots, S_4 and each party P_i (eventually) inputs its private $\text{ExternalValidity}_i$ as defined in Algorithm 5.5.3. The functionality is parameterized by the linear circuits L_1, \dots, L_n , each is defined as in Circuit 5.5.2.
 4. Each party P_i : upon receiving an output OK from Functionality 5.5.4, then terminate and output OK.
-

Theorem 5.5.6. *Let $n \geq 4t + 1$. Protocol 5.5.5, Π_{ProdVer} , securely computes $\mathcal{F}_{\text{ProdVer}}$ (Functionality 5.5.1) in the presence of a malicious adversary controlling at most t parties. It requires communication of $\mathcal{O}(n^3 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits of broadcast. Each party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

Proof. We show separately the case of an honest dealer and of a corrupted dealer.

The case of an honest dealer.

1. The simulator invokes the adversary \mathcal{A} .

2. The simulator receives from the functionality the shares $A^u(i), B^u(i), C^u(i)$ for every $i \in I$.
3. It chooses trivariate polynomials $\mathbf{S}_1(\mathbf{x}, \mathbf{y}, \mathbf{z}), \dots, \mathbf{S}_4(\mathbf{x}, \mathbf{y}, \mathbf{z})$ uniformly at random under the constraints that for every $i \in I$:

$$\sum_{r=1}^4 \sum_{k=1}^{t/2} i^{(r-1) \cdot (t/2) + k} \cdot \mathbf{S}_r(-\beta, -k, -\gamma) = A^{(\beta, \gamma)}(i) \cdot B^{(\beta, \gamma)}(i) - C^{(\beta, \gamma)}(i) .$$

Note that this imposes $(t/2 + 1)^2 \cdot |I| \leq (t/2 + 1)^2 \cdot t$ total constraints, while in each polynomial we pack up to $(t/2 + 1)^3$ secrets.

4. Simulate the invocation of Functionality 5.5.4: for every $i \in I$, give the adversary

$$\text{share}_i = ((\mathbf{S}_r(\mathbf{x}, \mathbf{y}, i), \mathbf{S}_r(\mathbf{x}, i, \mathbf{z}), \mathbf{S}_r(\mathbf{x}, \mathbf{y}, i))_{r \in [4]}, L_i(\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4))$$

5. It simulates the functionality returning OK. Whenever the adversary delivers the message OK to party P_j in the simulated protocol, the simulator delivers the output of Functionality 5.5.1 to party P_j in the ideal world.

We first show that the output of the honest parties is identical in the real and ideal executions. In the real execution, since the dealer is honest, it always holds valid polynomials $A^{(\beta, \gamma)}(\mathbf{x}), B^{(\beta, \gamma)}(\mathbf{x}), C^{(\beta, \gamma)}(\mathbf{x})$ each of degree- t such that $A^{(\beta, \gamma)}(0) \cdot B^{(\beta, \gamma)}(0) = C^{(\beta, \gamma)}(0)$ holds for every $(\beta, \gamma) \in V \times V$. Moreover, it chooses the trivariate polynomials $\mathbf{S}_1, \dots, \mathbf{S}_4$ such that they satisfy the conditions of Functionality 5.5.4. Specifically, $\mathbf{S}_1, \dots, \mathbf{S}_4$ are of the degree $t + t/2$ in each of the three variables and $\text{ExternalValidity}_j(\text{share}_j) = 1$ holds for all the honest parties. Hence, the functionality always returns OK to all the parties. Consequently, honest parties always output OK in the real execution. In the ideal execution, all honest parties hold shares on the valid degree- t polynomials $A^{(\beta, \gamma)}(\mathbf{x}), B^{(\beta, \gamma)}(\mathbf{x}), C^{(\beta, \gamma)}(\mathbf{x})$ for every $(\beta, \gamma) \in V \times V$ shared by the dealer, with which they invoke the Functionality 5.5.1. Since the dealer is honest, for each $(\beta, \gamma) \in V \times V$ it holds that $A^{(\beta, \gamma)}(0) \cdot B^{(\beta, \gamma)}(0) = C^{(\beta, \gamma)}(0)$. The output of the functionality is fully determined by the honest parties' inputs and hence it always returns OK to all the honest parties. It thus remains to show that the view of the adversary is identically distributed in the real and ideal executions.

Below, we prove the claim for a single trivariate polynomial to ease the notations. However, the claim easily extends to the case of four trivariate polynomials as used in the actual protocol.

Claim 5.5.7. *Let E_1 and E_2 be any arbitrary sets of $(t/2 + 1) \times t/2 \times (t/2 + 1)$ field elements each*

and let $I \subset [n]$ be a set of cardinality at most t . Then, under the constraint that for each $i \in I$ and $(\beta, \gamma) \in V \times V$, $\sum_{k=1}^{t/2} i^k \cdot E_1(\beta, k, \gamma) = \sum_{k=1}^{t/2} i^k \cdot E_2(\beta, k, \gamma)$, the following two distributions are identical:

Process I: Choose a random trivariate polynomial $\mathbf{F}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ such that $\mathbf{F}(-\beta, -k, -\gamma) = E_1(\beta, k, \gamma)$ for every $\beta, \gamma \in \{0, \dots, t/2\}$ and every $k \in \{1, \dots, t/2\}$. Output $(i, \mathbf{F}(\mathbf{x}, \mathbf{y}, i), \mathbf{F}(\mathbf{x}, i, \mathbf{z}), \mathbf{F}(i, \mathbf{y}, \mathbf{z}), \sum_{k=1}^{t/2} i^k \cdot \mathbf{F}(\mathbf{x}, -k, \mathbf{z}))$ for every $i \in I$.

Process II: Choose a random trivariate polynomial $\mathbf{F}'(\mathbf{x}, \mathbf{y}, \mathbf{z})$ such that $\mathbf{F}'(-\beta, -k, -\gamma) = E_2(\beta, k, \gamma)$ for every $\beta, \gamma \in \{0, \dots, t/2\}$ and every $k \in \{1, \dots, t/2\}$. Output $(i, \mathbf{F}'(\mathbf{x}, \mathbf{y}, i), \mathbf{F}'(\mathbf{x}, i, \mathbf{z}), \mathbf{F}'(i, \mathbf{y}, \mathbf{z}), \sum_{k=1}^{t/2} i^k \cdot \mathbf{F}'(\mathbf{x}, -k, \mathbf{z}))$ for every $i \in I$.

Proof. We show that the probability distributions $\{(i, \mathbf{F}(\mathbf{x}, \mathbf{y}, i), \mathbf{F}(\mathbf{x}, i, \mathbf{z}), \mathbf{F}(i, \mathbf{y}, \mathbf{z}), T_i(\mathbf{x}, \mathbf{z}))\}_{i \in I}$ corresponding to **Process I** and $\{(i, \mathbf{F}'(\mathbf{x}, \mathbf{y}, i), \mathbf{F}'(\mathbf{x}, i, \mathbf{z}), \mathbf{F}'(i, \mathbf{y}, \mathbf{z}), T'_i(\mathbf{x}, \mathbf{z}))\}_{i \in I}$ corresponding to **Process II** are identical. Towards that, let \mathbb{S} and \mathbb{S}' denote the probability ensembles corresponding to **Process I** and **Process II** respectively. We thus show that $\mathbb{S} \equiv \mathbb{S}'$. For this, we show that given any set of tuple of bivariate polynomials with degree $3t/2$ in both variables, say $Z = \{Q_i(\mathbf{x}, \mathbf{y}), W_i(\mathbf{x}, \mathbf{z}), R_i(\mathbf{y}, \mathbf{z}), T_i(\mathbf{x}, \mathbf{z})\}_{i \in I}$ that satisfy Definition 5.5.8, the number of trivariate polynomials in support of \mathbb{S} that are consistent with Z are the same as the number of polynomials in support of \mathbb{S}' .

Note that if the set Z does not satisfy Definition 5.5.8, then there does not exist any trivariate polynomial that is in support of \mathbb{S} or \mathbb{S}' . Now, consider a set Z that satisfies Definition 5.5.8. For simplicity, consider the case when $|I| = t$. Choose a set, say E , of $(t/2 + 1)^2$ elements selected uniformly at random from \mathbb{F} . Note that $E_1 \cup E$ together with Z defines a unique trivariate polynomial $S(\mathbf{x}, \mathbf{y}, \mathbf{z})$ as follows. For each $k \in \{0, \dots, t/2\}$, construct $W_{-k}(\mathbf{x}, \mathbf{z})$ of degree $-3t/2$ in each variable such that $W_0(-\beta, -\gamma) = E(\beta, \gamma)$ and $W_{-k}(-\beta, -\gamma) = E_1(\beta, k, \gamma)$. Note that this defines $(t/2 + 1)^2$ points on each $W_{-k}(\mathbf{x}, \mathbf{y})$. Moreover, for every $i \in I$, we set $W_{-k}(\mathbf{x}, i) = Q_i(\mathbf{x}, -k)$ and $W_{-k}(i, \mathbf{z}) = R_i(-k, \mathbf{z})$. This defines $2t(t + 1)$ more points on each $W_{-k}(\mathbf{x}, \mathbf{z})$. Thus, in total we have $(t + t/2 + 1)^2$ points defined on each $W_{-k}(\mathbf{x}, \mathbf{z})$ which defines the polynomial completely. Given the t polynomials $W_i(\mathbf{x}, \mathbf{z})$ for every $i \in I$ and $t/2 + 1$ polynomials $W_{-k}(\mathbf{x}, \mathbf{y})$ for every $k \in \{0, \dots, t/2\}$, these define a unique polynomial $\mathbf{F}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ due to Claim 5.3.7. The same argument holds true for the case of the set $E_2 \cup E$ and the corresponding unique polynomial $\mathbf{F}'(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Hence, the support for \mathbb{S} and \mathbb{S}' is equal. For the case when $|I| < t$, we can view process I (respectively process II) as first choosing $t - |I|$ polynomials $W_j(\mathbf{x}, \mathbf{z})$ (respectively $W'_j(\mathbf{x}, \mathbf{z})$) for some $j \notin I$ uniformly at random under the constraint that Z together with $\{W_j(\mathbf{x}, \mathbf{z})\}$ (respectively $\{W'_j(\mathbf{x}, \mathbf{z})\}$) satisfy Definition 5.5.8. Following this, the analysis is same as the case with $|I| = t$. \square

The case of a corrupted dealer.

1. The simulator invokes the adversary \mathcal{A} .
2. The simulator receives from the functionality the polynomials $A^{(\beta,\gamma)}(\mathbf{x})$, $B^{(\beta,\gamma)}(\mathbf{x})$ and $C^{(\beta,\gamma)}(\mathbf{x})$ for every $(\beta, \gamma) \in V \times V$.
3. It simulates the invocation of Functionality 5.5.4: It receives from the adversary four trivariate polynomials $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4$.
4. It generates the external validity functions $\text{ExternalValidity}_j$ for every $j \notin I$ with $a_j^{(\beta,\gamma)} = A^{(\beta,\gamma)}(j)$, $b_j^{(\beta,\gamma)} = B^{(\beta,\gamma)}(j)$ and $c_j^{(\beta,\gamma)} = C^{(\beta,\gamma)}(j)$. It sends to the adversary the predicates $\text{ExternalValidity}_j$ for every $j \notin I$ as in Functionality 5.5.4.
5. Define $T_i(\mathbf{x}, \mathbf{z}) = L_i(\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4)$ where L_i is defined as in Circuit 5.5.2.
6. Check that each polynomial \mathbf{S}_r for $r \in [4]$ is of degree $t + t/2$ in each one of the three variables. Moreover, if $\text{ExternalValidity}_j$ holds for at least $2t + 1$ honest parties, then simulate Functionality 5.5.4 sending output OK. Otherwise, do not terminate.
7. Whenever the adversary delivers the message OK to party P_j in the simulated protocol, the simulator delivers the output of Functionality 5.5.1 to party P_j in the ideal world.

Clearly the view of the adversary is the same in both executions. We now prove that if Functionality 5.5.4 returns OK, then Functionality 5.5.1 also return OK. That is, we claim that once \mathcal{A} sent polynomials \mathbf{S}_r for $r \in [4]$ for which $\text{ExternalValidity}_j$ holds for at least $2t + 1$ parties, then it holds that $A^{(\beta,\gamma)}(0) \cdot B^{(\beta,\gamma)}(0) = C^{(\beta,\gamma)}(0)$ for every $(\beta, \gamma) \in V \times V$. To see that, fix some (β, γ) . Consider the following polynomial:

$$\begin{aligned} Y_{(\beta,\gamma)}(\mathbf{x}) &:= \sum_{r=1}^4 \sum_{k=1}^{t/2} \mathbf{x}^{(r-1) \cdot (t/2) + k} \cdot \mathbf{S}_r(-\beta, -k, -\gamma) \\ &= e_1 \mathbf{x} + \dots + e_{2t} \mathbf{x}^{2t} \end{aligned}$$

where each e_ℓ directly corresponds to some $\mathbf{S}_r(-\beta, -k, -\gamma)$. This is a univariate polynomial where the only variable is \mathbf{x} and is of degree $2t$. Moreover, $Y_{(\beta,\gamma)}(j) = T_j(-\beta, -\gamma)$, where $T_j(\mathbf{x}, \mathbf{z}) = L_j(\mathbf{S}_1, \dots, \mathbf{S}_4)$ as defined in Step 5 of the simulation. Furthermore, consider the polynomial:

$$E^{(\beta,\gamma)}(\mathbf{x}) = A^{(\beta,\gamma)}(\mathbf{x}) \cdot B^{(\beta,\gamma)}(\mathbf{x}) - C^{(\beta,\gamma)}(\mathbf{x}) .$$

This is also a univariate polynomial of degree- $2t$. From the external validity property, for at least $2t + 1$ honest parties $J \subseteq [n]$ it holds that

$$E^{(\beta,\gamma)}(j) = a_j^{(\beta,\gamma)} \cdot b_j^{(\beta,\gamma)} - c_j^{(\beta,\gamma)} = T_j(-\beta, -\gamma) = Y_{(\beta,\gamma)}(j) .$$

Therefore, the two degree- $2t$ polynomials $E^{(\beta,\gamma)}(\mathbf{x}), Y_{(\beta,\gamma)}(\mathbf{x})$ agree. Since the constant term of the polynomial $Y_{(\beta,\gamma)}(\mathbf{x})$ is 0, we get that the constant term of $E^{(\beta,\gamma)}(\mathbf{x})$ is 0. Therefore it holds that

$$E^{(\beta,\gamma)}(0) = A^{(\beta,\gamma)}(0) \cdot B^{(\beta,\gamma)}(0) - C^{(\beta,\gamma)}(0) = 0,$$

and therefore $A^{(\beta,\gamma)}(0) \cdot B^{(\beta,\gamma)}(0) = C^{(\beta,\gamma)}(0)$ for every $(\beta, \gamma) \in V \times V$. \square

5.5.3 Trivariate Polynomial Verification – Protocol

In the remainder of this sub-section, we show how to implement Functionality 5.5.4. To ease notations, we show how to implement the functionality with general linear functions L_1, \dots, L_n and general $\text{ExternalValidity}_j$. Moreover, we assume that the dealer sends just one trivariate polynomial \mathbf{S} instead of four; generalizing for the case of four polynomial is straightforward. This construction is essentially our asynchronous weak-binding trivariate secret sharing, for which we provided an extensive overview in Section 5.2.2.

Definition 5.5.8. We say that the share that P_i received from the dealer

$$\begin{aligned} \text{share}_i &= (Q_i(\mathbf{x}, \mathbf{y}), W_i(\mathbf{x}, \mathbf{z}), R_i(\mathbf{y}, \mathbf{z}), T_i(\mathbf{x}, \mathbf{z})) \\ &= (\mathbf{S}(\mathbf{x}, \mathbf{y}, i), \mathbf{S}(\mathbf{x}, i, \mathbf{z}), \mathbf{S}(i, \mathbf{y}, \mathbf{z}), L_i(\mathbf{S}(\mathbf{x}, \mathbf{y}, \mathbf{z}))) \end{aligned}$$

is consistent with an exchange sub-share message $m_{j \rightarrow i}$ that P_j sends to P_i ,

$$m_{j \rightarrow i} := (\text{exchange}, j, i, f_i^{Q_j}(\mathbf{x}), g_i^{Q_j}(\mathbf{y}), f_i^{W_j}(\mathbf{x}), g_i^{W_j}(\mathbf{z}), f_i^{R_j}(\mathbf{y}), g_i^{R_j}(\mathbf{z}), t^{L_i(Q_j)}(\mathbf{x})),$$

denoted as $\text{consistent}(\text{share}_i, m_{j \rightarrow i}) = 1$, if the following conditions hold:

$$\begin{aligned} f_i^{Q_j}(\mathbf{x}) &= W_i(\mathbf{x}, j) & (= \mathbf{S}(\mathbf{x}, i, j)), & & g_i^{Q_j}(\mathbf{y}) &= R_i(\mathbf{y}, j) & (= \mathbf{S}(i, \mathbf{y}, j)), \\ f_i^{W_j}(\mathbf{x}) &= Q_i(\mathbf{x}, j) & (= \mathbf{S}(\mathbf{x}, j, i)), & & g_i^{W_j}(\mathbf{z}) &= R_i(j, \mathbf{z}) & (= \mathbf{S}(i, j, \mathbf{z})), \\ f_i^{R_j}(\mathbf{y}) &= Q_i(j, \mathbf{y}) & (= \mathbf{S}(j, \mathbf{y}, i)), & & g_i^{R_j}(\mathbf{z}) &= W_i(j, \mathbf{z}) & (= \mathbf{S}(j, i, \mathbf{z})), \end{aligned}$$

and,

$$t^{L_i(Q_j)}(\mathbf{x}) = T_i(\mathbf{x}, j)$$

Protocol 5.5.9: Trivariate Polynomial Verification – Π_{TriVer}

Input: The input of the dealer is some trivariate polynomial $\mathbf{S}(\mathbf{x}, \mathbf{y}, \mathbf{z})$. The input of each party P_j is some predicate $\text{ExternalValidity}_j$.

Public parameters: The protocol is parameterized by linear circuits L_1, \dots, L_n .

The protocol:

1. **(Share Distribution)** For each party P_i , the dealer sends the share:

$$(\text{share}, i, \mathbf{S}(\mathbf{x}, \mathbf{y}, i), \mathbf{S}(\mathbf{x}, i, \mathbf{z}), \mathbf{S}(i, \mathbf{y}, \mathbf{z}), L_i(\mathbf{S}(\mathbf{x}, \mathbf{y}, \mathbf{z})))$$

2. **(Exchange sub-share)** Each party P_i :

- (a) Upon receiving $(\text{share}, i, Q_i(\mathbf{x}, \mathbf{y}), W_i(\mathbf{x}, \mathbf{z}), R_i(\mathbf{y}, \mathbf{z}), T_i(\mathbf{x}, \mathbf{z}))$ from the dealer, check if:
 $\text{ExternalValidity}_i(Q_i(\mathbf{x}, \mathbf{y}), W_i(\mathbf{x}, \mathbf{z}), R_i(\mathbf{y}, \mathbf{z}), T_i(\mathbf{x}, \mathbf{z})) = 1$.
- (b) If the above condition holds, then for every P_j define the following seven polynomials:

$$\begin{aligned} f_j^{Q_i}(\mathbf{x}) &\stackrel{\text{def}}{=} Q_i(\mathbf{x}, j) \quad (= \mathbf{S}(\mathbf{x}, j, i)), & g_j^{Q_i}(\mathbf{y}) &\stackrel{\text{def}}{=} Q_i(j, \mathbf{y}) \quad (= \mathbf{S}(j, \mathbf{y}, i)), \\ f_j^{W_i}(\mathbf{x}) &\stackrel{\text{def}}{=} W_i(\mathbf{x}, j) \quad (= \mathbf{S}(\mathbf{x}, i, j)), & g_j^{W_i}(\mathbf{z}) &\stackrel{\text{def}}{=} W_i(j, \mathbf{z}) \quad (= \mathbf{S}(j, i, \mathbf{z})), \\ f_j^{R_i}(\mathbf{y}) &\stackrel{\text{def}}{=} R_i(\mathbf{y}, j) \quad (= \mathbf{S}(i, \mathbf{y}, j)), & g_j^{R_i}(\mathbf{z}) &\stackrel{\text{def}}{=} R_i(j, \mathbf{z}) \quad (= \mathbf{S}(i, j, \mathbf{z})), \end{aligned}$$

and

$$t^{L_j(Q_i)}(\mathbf{x}) = L_j(Q_i(\mathbf{x}, \mathbf{y})).$$

Then, define the message:

$$m_{i \rightarrow j} := \left(\text{exchange}, i, j, f_j^{Q_i}(\mathbf{x}), g_j^{Q_i}(\mathbf{y}), f_j^{R_i}(\mathbf{x}), g_j^{R_i}(\mathbf{z}), f_j^{W_i}(\mathbf{y}), g_j^{W_i}(\mathbf{z}), t^{L_j(Q_i)}(\mathbf{x}) \right).$$

- (c) Verify that $\text{consistent}(\text{share}_i, m_{i \rightarrow i}) = 1$ (as per Definition 5.5.8), i.e., the share that P_i received from the dealer is consistent with itself.
- (d) If all the above conditions hold, then P_i sends to each P_j its sub-share $m_{i \rightarrow j}$.
- (e) Upon receiving a message $m_{j \rightarrow i}$ from P_j , verify that it is consistent with share_i (i.e., $\text{consistent}(\text{share}_i, m_{j \rightarrow i}) = 1$), where share_i received from the dealer and consistent is as Definition 5.5.8. If so, then broadcast $\text{Good}(i, j)$.
3. **(Identifying Star or Clique)** The dealer does as follows. Initialize a dynamic undirected graph $G = (V, E)$ with $V = [n]$. Upon receiving broadcasted messages $\text{Good}(i, j)$ from P_i and $\text{Good}(j, i)$ from P_j , add the edge (i, j) to E . Run Algorithm 5.5.10 and if the output is not \perp then broadcast the output and output OK. Otherwise, continue to listen to Good messages and repeat.
4. **(Verifying Star or Clique)** Each party P_i :
- (a) Initialize an undirected graph $G_i = (V_i, E_i)$ with $V_i = [n]$. Upon receiving broadcasted messages $\text{Good}(i, j)$ from P_i and $\text{Good}(j, i)$ from P_j , add the edge (i, j) to E_i .

- i. If (Dense, C) is received from the broadcast of the dealer, validate that $|C| \geq n - t$, and that each node $i \in C$ has a degree at least $3t + t/2 + 1$ in G_i . If the conditions hold, output OK.
 - ii. If (BigStar, C, D) is received from the broadcast of the dealer, P_i verifies that $C \subset D$, $|C| \geq 2t + t/2 + 1$, $|D| \geq n - t$ and that for every $c \in C$ and $d \in D$ the edge (c, d) is in G_i . If the conditions hold, then output OK.
- (b) Otherwise, continue to listen to Good messages, and with each message it updates the graph G_i and repeats the above checks.
-

Algorithm 5.5.10: Finding a BigStar or a Clique

- **Input:** An undirected graph G over $[n]$.
 1. Initialize a set $C = \emptyset$.
 2. For each node i that has degree higher than $3t + t/2 + 1$, add i to C .
 3. If $|C| \geq 3t + 1$ then output (Dense, C).
 4. Otherwise, let $\bar{C} = [n] \setminus C$, i.e., the set of all nodes with degree less than $3t + t/2 + 1$. For each node $i \in \bar{C}$:
 - (a) Consider the graph $G[\Gamma(i)]$ which consists of all vertices in G that have an edge to i (including i). If $G[\Gamma(i)]$ consists of less than $3t + t/2 + 1$ vertices, then add arbitrary vertices in G (say the lexicographically first one) to have a graph with exactly $3t + t/2 + 1$ vertices.
 - (b) Run STAR algorithm on input $(G[\Gamma(i)], n', t/2)$ where n' is the number of vertices in $G[\Gamma(i)]$ (i.e., at least $3t + t/2 + 1$). If the output is (C, D), then output (BigStar, C, D).
 5. Otherwise, output \perp .
-

Theorem 5.5.11. *Let $n \geq 4t + 1$. Protocol 5.5.9, Π_{TriVer} , securely computes $\mathcal{F}_{\text{TriVer}}$ (Functionality 5.5.4) in the presence of a malicious adversary controlling at most t parties. It has a communication complexity of $\mathcal{O}(n^3 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits of broadcast. Each party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

To show the full simulatability, We prove the following two claims regarding the protocol: validity and binding.

Claim 5.5.12 (Validity). *If the dealer is honest and starts with $S(x, y, z)$, and the inputs of the honest parties $\text{ExternalValidity}_j$ are such that $\text{ExternalValidity}(\text{share}_j) = 1$ where share_j is defined as in the protocol, then the protocol eventually terminates and each honest party outputs OK.*

Proof. Since for each honest party it holds that $\text{ExternalValidity}(\text{share}_j) = 1$, and all shares of honest parties agree with each other, we have that eventually, each honest P_j will broadcast $\text{Good}(j, \ell)$ for all honest parties $\ell \notin I$. We show that the dealer must eventually broadcast either Dense or BigStar messages. Consider time T in which all Good messages between pairs of honest parties were received by the dealer. There are two cases to consider:

- The dealer broadcasted (Dense, C) or (BigStar, C, D) before time T ;
- At time T , if each honest node has degree at least $3t + t/2 + 1$ then we have a total of $n - t$ nodes that have high degree, and thus the dealer broadcasts (Dense, C) where C contains (at least) all honest parties.
- Otherwise, there exists an honest party $j \notin I$ that has degree smaller than $3t + t/2 + 1$. Since we are at time T , all the missing edges are of corrupted parties. That is, when considering the graph $G[\Gamma(j)]$, the vertices that are removed correspond to $t/2$ corrupted parties. Thus, the graph $G[\Gamma(j)]$ contains $n' \geq 3t + t/2 + 1$ vertices, and contains a clique of size $3t + 1$ (i.e., all honest parties). According to Claim 5.3.5, the STAR algorithm finds a (C, D)-star with $|C| \geq n' - 2 \cdot (t/2) \geq 2t + t/2 + 1$ and $|D| \geq n' - t/2 \geq 3t + 1$.

The dealer thus eventually broadcasts one of the messages (Dense, C) or (BigStar, C, D), and all honest parties eventually receive this message. Moreover, since all Good messages are broadcasted, eventually all honest parties will see the same edges as the honest dealer, and validates the (Dense, C) or (BigStar, C, D) messages. Once the broadcasted message of the dealer is validated by an honest P_j , it halts and output OK. \square

Claim 5.5.13 (Termination). *If one honest parties terminate, then all honest parties eventually terminates.*

Proof. This follows immediately from the guarantees of the broadcast (aka. A-cast): If the dealer broadcasts (Dense, C) or (BigStar, C, D) then all honest parties will eventually see this message. Moreover, if the property holds in the graph of one honest party, then all honest parties eventually see those edges and validate the property in their respective graphs. This is because the graph is defined by the Good messages that were also broadcasted. \square

Claim 5.5.14 (Binding). *When the first honest party terminates, there exists a unique trivariate polynomial $S(x, y, z)$ of degree $t + t/2$ in each one of the variables x, y, z that can be extracted*

from the views of the honest parties. Moreover, for at least $2t + 1$ honest parties $J \subset [n] \setminus I$ it holds that $\text{ExternalValidity}_j(\text{share}_j) = 1$.

Proof. Consider the first honest party that terminates, say some P_{j^*} . If P_{j^*} terminates then the dealer must have broadcasted Dense or BigStar messages, and P_{j^*} validated the respective property in its local graph. There are two cases to consider:

Case I: The property is (Dense, C). In that case, the graph that the honest party P_{j^*} sees contains $n - t$ vertices C , where each has degree at least $3t + t/2 + 1$. Consider the set of the lexicographically first $t + t/2 + 1$ honest parties $H \subseteq C \setminus I$. Each such party inputs some $Q_h(\mathbf{x}, \mathbf{y})$, $W_h(\mathbf{x}, \mathbf{z})$ and $R_h(\mathbf{y}, \mathbf{z})$. Consider the unique trivariate polynomial $\mathbf{S}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ that satisfies $\mathbf{S}(\mathbf{x}, \mathbf{y}, h) = Q_h(\mathbf{x}, \mathbf{y})$ for every $h \in H$. Such a trivariate polynomial is guaranteed to exist by Claim 5.3.7.

We now show that all honest parties in C agree with $\mathbf{S}(\mathbf{x}, \mathbf{y}, \mathbf{z})$.

Claim 5.5.15. *For every honest party $j \in C$ it holds that $R_j(\mathbf{y}, \mathbf{z}) = \mathbf{S}(j, \mathbf{y}, \mathbf{z})$.*

Proof. Fix some $j \in C$ and $h \in H$. Since P_j and P_h both have bivariate polynomials with degrees $3t + t/2 + 1$ (otherwise, they would never send Good), and so they have at least $3t + 1$ vertices in common, in which at least $2t + 1$ honest parties are in their intersection. That is, there is a set K of honest parties of cardinality at least $2t + 1$ in which $\text{Good}(j, k)$ and $\text{Good}(h, k)$ was broadcasted for every $k \in K$. In particular, P_h and P_k verified that (among other things):

$$(\mathbf{S}(\mathbf{x}, k, h) =) \quad Q_h(\mathbf{x}, k) = f_k^{Q_h}(\mathbf{x}) = f_h^{W_k}(\mathbf{x}) = W_k(\mathbf{x}, h) .$$

P_k and P_j also exchanged shares, and verified that $g_j^{W_k}(\mathbf{z}) = g_k^{R_j}(\mathbf{z})$, that is, it holds that

$$W_k(j, \mathbf{z}) = g_j^{W_k}(\mathbf{z}) = g_k^{R_j}(\mathbf{z}) = R_j(k, \mathbf{z}) ,$$

and in particular on $\mathbf{z} = h$:

$$\mathbf{S}(j, k, h) = W_k(j, h) = R_j(k, h) .$$

Since this holds for every $k \in K$, we get that two degree $t + t/2$ polynomials $\mathbf{S}(j, \mathbf{y}, h)$ and $R_j(\mathbf{y}, h)$ agree on $2t + 1$ points, and therefore $\mathbf{S}(j, \mathbf{y}, h) = R_j(\mathbf{y}, h)$. Moreover, since this holds for every $h \in H$, we have that two degree- $(t + t/2)$ bivariate polynomials $\mathbf{S}(j, \mathbf{y}, \mathbf{z})$ and $R_j(\mathbf{y}, \mathbf{z})$ must agree, i.e., $\mathbf{S}(j, \mathbf{y}, \mathbf{z}) = R_j(\mathbf{y}, \mathbf{z})$. \square

Claim 5.5.16. For every honest party $j \in C$ it holds that $W_j(\mathbf{x}, \mathbf{z}) = \mathbf{S}(\mathbf{x}, j, \mathbf{z})$.

Proof. As before, fix some $j \in C$ and $h \in H$, and consider K of cardinality $2t + 1$ that agree both with P_j and P_h . For every $k \in K$, P_h and P_k verified that

$$(\mathbf{S}(k, \mathbf{y}, h) =) \quad Q_h(k, \mathbf{y}) = g_k^{Q_h}(\mathbf{y}) = f_h^{R_k}(\mathbf{y}) = R_k(\mathbf{y}, h) .$$

Moreover, P_j and P_k verified that

$$W_j(k, \mathbf{z}) = g_k^{W_j}(\mathbf{z}) = g_j^{R_k}(\mathbf{z}) = R_k(j, \mathbf{z}) .$$

In particular, for $\mathbf{z} = h$ it holds that $W_j(k, h) = R_k(j, h) = \mathbf{S}(k, j, h)$. Since this holds for every $k \in K$, we have that two univariate polynomials $W_j(\mathbf{x}, h)$ and $\mathbf{S}(\mathbf{x}, j, h)$ must agree. Since this holds for every $h \in H$, we get that the two polynomials $W_j(\mathbf{x}, \mathbf{y})$ and $\mathbf{S}(\mathbf{x}, j, \mathbf{y})$ must agree, and so $W_j(\mathbf{x}, \mathbf{y}) = \mathbf{S}(\mathbf{x}, j, \mathbf{y})$. \square

Claim 5.5.17. For every honest party $j \in C$ it holds that $Q_j(\mathbf{x}, \mathbf{y}) = \mathbf{S}(\mathbf{x}, \mathbf{y}, j)$.

Proof. As before, P_h and P_k exchanged the shares

$$(\mathbf{S}(h, k, \mathbf{z}) =) \quad R_h(k, \mathbf{z}) = g_k^{R_h}(\mathbf{z}) = g_h^{W_k}(\mathbf{z}) = W_k(h, \mathbf{z}) ,$$

where $\mathbf{S}(h, k, \mathbf{z}) = R_h(k, \mathbf{z})$ follows from Claim 5.5.15. Moreover, P_k and P_j verified that

$$Q_j(\mathbf{x}, k) = f_k^{Q_j}(\mathbf{x}) = f_j^{W_k}(\mathbf{x}) = W_k(\mathbf{x}, j) ,$$

and in particular for $\mathbf{x} = h$ it holds that $Q_j(h, k) = W_k(h, j) = \mathbf{S}(h, k, j)$. Since this holds for every $k \in K$, we get that the two univariate polynomials $Q_j(h, \mathbf{y})$ and $\mathbf{S}(h, \mathbf{y}, j)$ agree. Since it also holds for every $h \in H$, this means that $Q_j(\mathbf{x}, \mathbf{y}) = \mathbf{S}(\mathbf{x}, \mathbf{y}, j)$. \square

Case II: The property is (BigStar, C, D). In that case, the graph that the honest party P_j^* sees contains a clique C of size $2t + t/2 + 1$. This implies that there is a set of at least $t + t/2 + 1$ honest parties for which for every $j, k \in C$, the party P_{j^*} viewed $\text{Good}(j, k)$ and $\text{Good}(k, j)$, and thus the shares that P_k and P_j agree with each other. Let H be the lexicographically first $t + t/2 + 1$ honest parties in C . Consider the unique trivariate polynomial $\mathbf{S}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ that satisfies $\mathbf{S}(\mathbf{x}, \mathbf{y}, h) = Q_h(\mathbf{x}, \mathbf{y})$ for every $h \in H$. Such a trivariate polynomial is guaranteed to exist by Claim 5.3.7. We now show that all honest parties in D hold shares on the polynomial $\mathbf{S}(x, y, z)$.

All honest parties in H hold shares on $\mathbf{S}(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Clearly, for every $h \in H$ it holds that $Q_h(\mathbf{x}, \mathbf{y}) = \mathbf{S}(\mathbf{x}, \mathbf{y}, h)$. For every $i, j \in H$ we have that P_i and P_j verified their shares, and thus we have that

$$\mathbf{S}(\mathbf{x}, i, j) = Q_j(\mathbf{x}, i) = f_i^{Q_j}(\mathbf{x}) = f_j^{W_i}(\mathbf{x}) = W_i(\mathbf{x}, j) .$$

Since this holds for every $j \in H$, we get that that the two $t + t/2 + 1$ bivariate polynomials $\mathbf{S}(\mathbf{x}, i, \mathbf{z})$ and $W_i(\mathbf{x}, \mathbf{z})$ agree, and so for every $i \in H$, $\mathbf{S}(\mathbf{x}, i, \mathbf{z}) = W_i(\mathbf{x}, \mathbf{z})$.

Similarly, for every $i, j \in H$ we have that P_i and P_j verified that

$$(\mathbf{S}(i, \mathbf{y}, j) =) \quad Q_j(i, \mathbf{y}) = g_i^{Q_j}(\mathbf{y}) = f_j^{R_i}(\mathbf{y}) = R_i(\mathbf{y}, j)$$

Since this holds for every $j \in H$, we get that the two $t + t/2 + 1$ bivariate polynomials $\mathbf{S}(i, \mathbf{y}, \mathbf{z})$ and $R_i(\mathbf{y}, \mathbf{z})$ agree on $t + t/2 + 1$ univariate polynomials, and so for every $i \in H$ it holds that $\mathbf{S}_i(\mathbf{y}, \mathbf{z}) = R_i(\mathbf{y}, \mathbf{z})$.

All honest parties in D hold shares on $\mathbf{S}(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Similarly to above, each party in $i \in D$ agrees with each party in H . Thus, for every $i \in D$ and $h \in H$ we have that

$$(\mathbf{S}(\mathbf{x}, h, i) =) \quad W_h(\mathbf{x}, i) = f_i^{W_h}(\mathbf{x}) = f_h^{Q_i}(\mathbf{x}) = Q_i(\mathbf{x}, h) .$$

Since this holds for every $h \in H$, we have that $\mathbf{S}(\mathbf{x}, \mathbf{y}, i) = Q_i(\mathbf{x}, \mathbf{y})$. Likewise,

$$\mathbf{S}(\mathbf{x}, i, h) = Q_h(\mathbf{x}, i) = f_i^{Q_h}(\mathbf{x}) = f_h^{W_i}(\mathbf{x}) = W_i(\mathbf{x}, h) .$$

Since this holds for every $h \in H$, we get that $W_i(\mathbf{x}, \mathbf{z}) = \mathbf{S}(\mathbf{x}, i, \mathbf{z})$. Finally,

$$(\mathbf{S}(i, \mathbf{y}, h) =) \quad Q_h(i, \mathbf{y}) = g_i^{Q_h}(\mathbf{y}) = f_h^{R_i}(\mathbf{y}) = R_i(\mathbf{y}, h) ,$$

and since this holds for every $h \in H$, we have that $\mathbf{S}(i, \mathbf{y}, \mathbf{z}) = R_i(\mathbf{y}, \mathbf{z})$.

External validity. An honest party P_j does not send shares to other parties, and in particular does not broadcast $\text{Good}(j, k)$ for every party P_k if its external validity was not 1. In the case of (Dense, C), we are guaranteed to have $|C| \geq n - t \geq 3t + 1$ and therefore the set contains at least $2t + 1$ honest parties. Since those parties broadcasted Good , we have $2t + 1$ honest parties with validated external validity predicate. Likewise, in the case of (BigStar, C, D), we have that $|D| \geq n - t \geq 3t + 1$, honest parties in D validated their external validity, and therefore we have at least $2t + 1$ honest parties with validated external validity predicate. \square

Proof. We are now ready to show the simulation.

The case of an honest dealer. In the honest dealer case, we assume that the honest parties input $\text{ExternalValidity}_j$ which on the shares of the honest parties output 1. We then have:

1. The simulator receives from the functionality the shares of the corrupted parties share_i for every $i \in I$.
2. Invoke the adversary \mathcal{A} . Simulate \mathcal{A} receiving headers (shares, j) for every $j \in [n]$. Moreover, simulate \mathcal{A} receiving the message $(\text{share}, i, \text{share}_i)$ where share_i is as received from the functionality.
3. Once \mathcal{A} delivers the message (shares, j) for $j \notin I$, simulate P_j sending $\text{exchange}(j, k)$ for every $k \notin I$. Moreover, for each $i \in I$, simulate P_j sending the message $m_{j \rightarrow i}$ to P_i , where

$$m_{j \rightarrow i} := (\text{exchange}, j, i, Q_i(\mathbf{x}, j), R_i(\mathbf{y}, j), Q_i(\mathbf{x}, j), R_i(j, \mathbf{z}), Q_i(j, \mathbf{y}), W_i(j, \mathbf{z}), T_i(\mathbf{x}, j)) .$$

4. For every message \mathcal{A} sends in the name of P_i to some honest P_j a message $m_{i \rightarrow j}$, check that $m_{i \rightarrow j}$ was sent correctly according to the protocol given share_i . Once \mathcal{A} delivers the message to P_j and the message is correct, simulate P_j broadcasting $\text{Good}(j, i)$.
5. Once \mathcal{A} delivers the message $\text{exchange}(j, k)$ from P_j to P_k (and it already delivered the message $\text{share}(k)$ from the dealer to P_k), then simulate P_k broadcasting $\text{Good}(k, j)$.
6. Simulate the honest dealer as in the protocol, running Algorithm 5.5.10. Since all honest parties eventually broadcast $\text{Good}(k, \ell)$ for every pair $k, \ell \notin I$, eventually (as shown in Claim 5.5.12) the dealer will ask to broadcast either (Dense, C) or $(\text{BigStar}, \text{C}, \text{D})$. Simulate the dealer broadcasting this message.
7. Simulate each honest party P_j verifying the validity of the broadcasted messages by the dealer. When the simulated P_j terminates (this occurs when the adversary delivers some message to P_j , either a message broadcasted by the dealer or some other party), then the simulator allows the deliver of the output of the functionality (which is OK) to the honest party P_j in the ideal world.

We now show that the view of the environment \mathcal{Z} is the same in both executions. From inspection, it is easy to see that the view of the adversary \mathcal{A} is the same in both executions. In particular, this is due to the fact that the protocol and the simulation are deterministic, and that the messages of the honest parties to the corrupted parties can be simulated from the shares of the corrupted parties. As follows from Claim 5.5.12, the output of the honest parties in the case of an honest dealer is always OK (assuming that the external validity predicates

that the honest parties input to the functionality give 1 on the shares provided by the dealer). Therefore, the output of the honest parties in the real world is always OK, the same as in the ideal world. The scheduling in which parties receive outputs is determined by the adversary (i.e., the controlling of the router), and since the view of the adversary is exactly the same in both executions, its control over the router is exactly the same.

The case of a corrupted dealer.

1. The simulator invokes the adversary \mathcal{A} .
2. The simulator receives from the functionality the external validity predicates, i.e., $\text{ExternalValidity}_j$, for every $j \notin I$.
3. The simulator simulates the honest parties in an execution of the protocol where the input of each P_j is $\text{ExternalValidity}_j$. The simulator also simulates the router of the real world. Note that the simulation might not terminate.
4. When the first honest party P_{j^*} terminates, all honest parties will eventually terminate, see Claim 5.5.14. The Claim also shows how to extract a trivariate polynomial $S(\mathbf{x}, \mathbf{y}, \mathbf{z})$ from the views of the honest parties. The simulator sends S to the trusted party. For this particular S , the external validity property holds for at least $2t + 1$ honest parties. Moreover, S is of degree at most $t + t/2$ in all three variables $\mathbf{x}, \mathbf{y}, \mathbf{z}$. Thus, the functionality will accept this polynomial, and it would send OK to all parties.
5. The simulator continues to simulate the protocol with the adversary. Whenever a simulated honest party P_j obtains an output in the simulated protocol, the simulator \mathcal{S} delivers to the router to allow sending the dummy party P_j in the ideal its output, OK.
6. Eventually, all honest parties will receive output in the simulated execution. The simulator then terminates.

The view of the adversary is exactly the same in the real and ideal executions, as the simulated honest parties are deterministic and use the exact same inputs as in the real world. If some honest party terminates, then as follows from Claim 5.5.14 all honest parties would eventually terminate and with their shares lie on the same polynomial $S(\mathbf{x}, \mathbf{y}, \mathbf{z})$. The environment \mathcal{Z} sees the outputs at the same activation in the real and ideal: Upon the activation in which the first honest party P_{j^*} terminates, the simulator extracts the trivariate polynomial and sends it to the trusted party, and then deliver the output to P_{j^*} . Since the view of the adversary is exactly the same, whenever a real honest party P_k receives an output, a simulated P_k receives an output in the simulation, and then the simulator extracts the router to deliver the output of the functionality to P_k in the ideal world. \square

5.6 Rate-1 Asynchronous Weak-Binding Secret Sharing

Protocol 5.5.9 provides a secret sharing of a trivariate polynomial with $\mathcal{O}(1)$ overhead. Along the way, it also allows some external verification (ExternalValidity) and some computation on the trivariate shares. This section describes our weak-binding secret-sharing protocol with a shunning reconstruction. We remark again that we do not use this protocol in our work. Nevertheless, we provide it as an independent primitive for completeness and as it might be useful as an independent primitive.

Definition 5.6.1 (Asynchronous Weak-binding Secret Sharing with Shunning Reconstruction). *Let S be a finite domain, $|S| \geq 2$, and let $[n]$ be a set of parties that includes a distinguished dealer. An asynchronous weak-binding secret sharing with shunning reconstruction consists of two phases, a sharing phase and a reconstruction phase, with the following syntax.*

- *Sharing: At the beginning, the dealer holds a secret $s \in S$ and each party including the dealer holds an independent random input r_i . The parties may communicate in several time in sequence. Each time, each party can privately send messages to the other parties and it can also broadcast a message. Each message sent or broadcasted by P_i is determined by the view of P_i , consists of its input (if any), its random input and messages received from other parties in previous rounds.*
- *Reconstruction: At the beginning of the reconstruction, the parties are holding their view from the sharing phase and in addition the dealer maintains a list L which is initialized to \emptyset . The reconstruction phase may involve several interactions, and at each time the parties send messages based on their view. At the end of the reconstruction, each party either outputs a value or never terminates. When the parties do not terminate, the dealer will have at least $t/2 + 1$ parties in its list L .*

We should have the following properties for any adversary $A = (A_s, A_r)$ corrupting at most t parties:

- *Termination: If the dealer is honest then each honest party eventually terminates the sharing phase. If some honest party terminates the sharing phase, then every honest party must terminate it eventually. For the reconstruction phase one of these must hold: (a) If some honest party terminates the reconstruction phase, then all other honest parties will eventually terminate or (b) the dealer will have at least $t/2 + 1$ parties in L .*

- **Privacy:** If D is honest then the adversary's view during the sharing phase reveals almost no information on s . Formally, let D_s is the view A in the sharing phase on secret s . Then, for any $s \neq s'$, the random variables D_s and $D_{s'}$ are identical.
- **Weak-binding:** At the end of the sharing phase there is a value $s^* \in S$ such that at the end of the reconstruction phase, if the parties terminate, then the output will be s^* . If the dealer is honest, then $s^* = s$.

Protocol 5.6.2: Asynchronous Weak-binding Secret Sharing – Π_{AWBSS}

Input: The input of the dealer is some trivariate polynomial $S(x, y, z)$. Each other party has no input.

Sharing phase:

1. Each party P_i and the dealer: Run Protocol 5.5.9 with $\text{ExternalValidity}_i(\cdot)$ as the predicate that always returns 1, and each $L_j(\cdot) = \perp$ for every share j .
2. If the protocol terminates with output OK, then:
 - (a) If (Dense, C) was received as the broadcasted message from the dealer: if $i \in C$ then store $X = C$ and share $_i$. (excluding the last element – which is \perp .)
 - (b) If (BigStar, C, D) was received as the broadcasted message from the dealer: if $i \in D$ then store $X = D$ and share $_i$. (excluding the last element.)
 - (c) Otherwise, store $X = C$ if Dense and $X = D$ if BigStar.

Shunning Reconstruction phase:

1. **(Broadcasting the Polynomial)** The dealer:
 - (a) Initialise a shunning list $\text{Shun} = X$.
 - (b) Broadcast a trivariate polynomial $S(x, y, z)$.
2. **(Verifying the Dealer's Polynomial)** Each party P_i :
 - (a) Upon receiving a polynomial $S(x, y, z)$ from the dealer, verify that the polynomial is of degree at most $t + t/2$ in each variable. If not, then discard the dealer and terminate.
 - (b) If $i \in X$, then verify that $S(x, y, i) = Q_i(x, y)$, $S(x, i, z) = W_i(x, z)$ and $S(i, y, z) = R_i(y, z)$ holds. If all the conditions hold, then P_i broadcasts OK.
3. **(Output)**
 - (a) Upon receiving the OK from the broadcast of P_i , the dealer updates $\text{Shun} = \text{Shun} \setminus \{i\}$.

- (b) Upon receiving OK from at least $2t + t/2 + 1$ parties in X , party P_i outputs $S(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and terminates. Otherwise, it continues to wait for OK messages.
-

We use the term “shunning” for reconstruction to indicate that the reconstruction phase offers the following guarantees: either the reconstruction succeeds, or the dealer can identify at least $t/2 + 1$ parties thereafter.

Essentially, in an honest dealer case, termination of the sharing phase is guaranteed. However, since the set X does not necessarily contain $2t + t/2 + 1$ honest parties, for reconstruction, we might need the adversary’s help. This is why the reconstruction is either guaranteed, or the dealer shuns at least $t/2 + 1$ parties. In the case of a corrupted dealer, once the sharing terminates, reconstruction must be to the same polynomial (or discard the dealer, or not terminate, but cannot be ended successfully with a different polynomial). We formalize and prove the properties of this protocol below.

Claim 5.6.3 (Sharing Termination). *If the dealer is honest, then each honest party terminates the sharing phase. If some honest party terminates the sharing phase then every honest party must terminate it eventually.*

Proof. Since for each honest party P_j , we have $\text{ExternalValidity}_j(\cdot) = 1$, for an honest dealer, by Claim 5.5.12 we have that Protocol 5.5.9 terminates with the output OK. Hence, sharing phase always completes successfully. The latter follows immediately from Claim 5.5.13. \square

Claim 5.6.4 (Reconstruction Termination). *Either all the honest parties terminate the reconstruction, or the dealer shuns at least $t/2 + 1$ parties.*

Proof. We have the following two cases to consider:

1. **There exists some honest party P_{j^*} which terminates:** This implies that P_{j^*} received at least $2t + t/2 + 1$ broadcasts of OK messages from the parties in set X identified during the sharing phase. These messages will be eventually received by all the honest parties (including the dealer), ensuring that all the honest parties terminate.
2. **No honest party has terminated:** This implies that less than $2t + t/2 + 1$ OK messages are received by the honest parties, which includes the dealer. In that case, the dealer’s shunning set consists of all the parties from X from whom the dealer has not received a broadcast of OK. Since $|X| \geq n - t$, we have that $|\text{Shun}| \geq t/2 + 1$.

\square

The following claim follows from Theorem 5.5.11:

Claim 5.6.5 (Privacy). *If the dealer is honest, then the adversary’s view during the sharing phase reveals no information on the dealer’s input.*

Claim 5.6.6 (Weak Binding). *At the termination of the sharing phase, there is a unique trivariate polynomial $S'(x, y, z)$ with degree $3t/2$ that might be reconstructed in the reconstruction phase. Moreover, if the dealer is honest then $S'(x, y, z) = S(x, y, z)$ where $S(x, y, z)$ is the input of the dealer.*

Proof. If the sharing phase terminates for the honest parties, it implies that Protocol 5.5.9 terminates with OK and all the parties hold a set X of size at least $n - t$. By Claim 5.5.14, we have that all the honest parties in X hold shares on some unique trivariate polynomial $S'(x, y, z)$.

Further, if the reconstruction terminates for some honest party, it implies that the dealer broadcasted a trivariate polynomial, say S^* , with degree $3t/2$ in each variable. Moreover, at least $2t + t/2 + 1$ parties from X broadcasted OK after verifying the consistency of their shares with the dealer’s broadcasted trivariate polynomial. Of these, at least $t + t/2 + 1$ parties are guaranteed to be honest. We have that the two polynomials S^* and S' agree in at least $(t + t/2 + 1)^3$ points, and hence $S^* = S'$. Moreover, an honest dealer always broadcasts $S^* = S$ and hence parties output the dealer’s polynomial. \square

5.7 Verifiable Triple Sharing

In this section, we build upon packed VSS (Functionality 5.4.1) and Functionality 5.5.1 to show how a dealer can verifiably share $\mathcal{O}(n^2)$ triples simultaneously.

The functionality for verifiable triple sharing appears below, followed by the protocol. The Shamir-shares of party P_j for the $(t/2 + 1)^2$ multiplication triples are as follows, following the invocation of the functionality or the protocol: $(A^u(-\beta, j), B^u(-\beta, j), C^u(-\beta, j))$ for every $u, \beta \in \{0, \dots, t/2\}$.

Functionality 5.7.1: Verifiable Triple Secret Sharing – $\mathcal{F}_{\text{PVTs}}$

The functionality is parameterized by a set of corrupted parties $I \subseteq [n]$.

1. The dealer sends to the functionality 3 sets of $t/2 + 1$ polynomials $\{A^u(x, y)\}$, $\{B^u(x, y)\}$ and $\{C^u(x, y)\}$ for each $u \in \{0, \dots, t/2\}$.
2. The functionality verifies that each polynomial is of degree at most $t + t/2$ in x and t in y . If not, it does not terminate.

3. If the dealer is honest, then for each $u \in \{0, \dots, t/2\}$, give adversary the shares $(A^u(\mathbf{x}, i), A^u(i, \mathbf{y}))$, $(B^u(\mathbf{x}, i), B^u(i, \mathbf{y}))$ and $(C^u(\mathbf{x}, i), C^u(i, \mathbf{y}))$ for every $i \in I$.
4. The functionality verifies that for every $u, \beta \in \{0, \dots, t/2\}$ it holds that

$$A^u(-\beta, 0) \cdot B^u(-\beta, 0) = C^u(-\beta, 0) .$$

If yes, then it sends $(A^u(\mathbf{x}, j), A^u(j, \mathbf{y}))$, $(B^u(\mathbf{x}, j), B^u(j, \mathbf{y}))$ and $(C^u(\mathbf{x}, j), C^u(j, \mathbf{y}))$ for every $u \in \{0, \dots, t/2\}$ to each party P_j and halts. Otherwise, the functionality never terminates.

Protocol 5.7.2: Verifiable Triple Secret Sharing Protocol – Π_{PVTs}

- **Input:** The dealer holds the polynomials $A^u(\mathbf{x}, \mathbf{y})$, $B^u(\mathbf{x}, \mathbf{y})$ and $C^u(\mathbf{x}, \mathbf{y})$ of degree $t + t/2$ in \mathbf{x} and t in \mathbf{y} for every $u \in \{0, \dots, t/2\}$ such that $A^u(-\beta, 0) \cdot B^u(-\beta, 0) = C^u(-\beta, 0)$ holds for every $\beta \in \{0, \dots, t/2\}$.
- **The protocol:**
 1. The dealer invokes Functionality 5.4.1 with its polynomials $A^u(\mathbf{x}, \mathbf{y})$, $B^u(\mathbf{x}, \mathbf{y})$ and $C^u(\mathbf{x}, \mathbf{y})$ for every $u \in \{0, \dots, t/2\}$ in a batched manner.
 2. The dealer invokes Functionality 5.5.1 with the input $A^u(-\beta, \mathbf{y})$, $B^u(-\beta, \mathbf{y})$ and $C^u(-\beta, \mathbf{y})$ for every $u, \beta \in \{0, \dots, t/2\}$.
 3. Upon receiving an output $(A^u(\mathbf{x}, j), A^u(j, \mathbf{y}))$, $(B^u(\mathbf{x}, j), B^u(j, \mathbf{y}))$ and $(C^u(\mathbf{x}, j), C^u(j, \mathbf{y}))$ from the functionality, each P_j invokes the Functionality 5.5.1 with the inputs $(A^u(-\beta, j), B^u(-\beta, j), C^u(-\beta, j))$ for every $u, \beta \in \{0, \dots, t/2\}$.
 4. Upon receiving an output OK from the Functionality 5.5.1, P_j outputs $(A^u(\mathbf{x}, j), A^u(j, \mathbf{y}))$, $(B^u(\mathbf{x}, j), B^u(j, \mathbf{y}))$ and $(C^u(\mathbf{x}, j), C^u(j, \mathbf{y}))$, where $(A^u(-\beta, j), B^u(-\beta, j), C^u(-\beta, j))$ for every $u, \beta \in \{0, \dots, t/2\}$ defines P_j 's degree- t Shamir-share of the $(t/2 + 1)^2$ multiplication triples.

Theorem 5.7.3. *Let $n \geq 4t + 1$. Protocol 5.7.2, Π_{PVTs} , securely computes $\mathcal{F}_{\text{PVTs}}$ (Functionality 5.7.1) in the presence of a malicious adversary controlling at most t parties. It has a communication complexity of $\mathcal{O}(n^3 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits of broadcast for sharing $\mathcal{O}(n^2)$ triples simultaneously. Each party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

Proof. We show the case of an honest dealer and of a corrupted dealer separately.

Case I – the case of an honest dealer. The simulator \mathcal{S} is as follows:

1. Upon activation, invoke the adversary \mathcal{A} .
2. The simulator receives from the functionality the shares $(f_i^{A^u}(\mathbf{x}), g_i^{A^u}(\mathbf{y}))$, $(f_i^{B^u}(\mathbf{x}), g_i^{B^u}(\mathbf{y}))$ and $(f_i^{C^u}(\mathbf{x}), g_i^{C^u}(\mathbf{y}))$ for every $u \in \{0, \dots, t/2\}$ and every $i \in I$.
3. Simulate the invocation of the inner Functionality 5.4.1 for the adversary: for every $i \in I$, send to the adversary the shares $(f_i^{A^u}(\mathbf{x}), g_i^{A^u}(\mathbf{y}))$, $(f_i^{B^u}(\mathbf{x}), g_i^{B^u}(\mathbf{y}))$ and $(f_i^{C^u}(\mathbf{x}), g_i^{C^u}(\mathbf{y}))$ for every $u \in \{0, \dots, t/2\}$.
4. Simulate the invocation of the inner Functionality 5.5.1 for the adversary: for every $i \in I$, give the adversary $(f_i^{A^u}(-\beta), f_i^{B^u}(-\beta), f_i^{C^u}(-\beta))$ for every $u, \beta \in \{0, \dots, t/2\}$.
5. Simulate the Functionality 5.5.1 returning OK. When the adversary delivers OK to a party P_j in the simulated protocol, the simulator delivers the output of Functionality 5.7.1 to P_j in the ideal world.

Clearly, since the protocol and the simulation are deterministic, the view of the adversary \mathcal{A} is identical in both the real and ideal executions. It thus remains to show that the output of honest parties is the same in both the executions.

In the ideal world, an honest dealer always invokes the functionality with valid polynomials. Hence, the functionality delivers the shares on the dealer's polynomials to each honest party. In the real world, an honest dealer's polynomials always satisfy the conditions of Functionality 5.4.1 and from its guarantees (Claim 5.4.5) we have that each honest party receives its share. Moreover, the dealer's polynomials also satisfy the conditions of Functionality 5.5.1. Hence the functionality returns OK to all the honest parties, which in turn output their respective shares on the dealer's polynomials. Hence, the output of honest parties is identical in the real and ideal executions. Moreover, although the scheduling of message delivery is determined by the adversary \mathcal{A} , its view is identical in both the executions. Hence, the scheduling is also identical.

Case II – the case of a corrupt dealer. The simulator \mathcal{S} is as follows:

1. The simulator invokes the adversary \mathcal{A} .
2. Since the protocol is deterministic and the honest parties do not have any input to the protocol, the simulator can simulate all honest parties in an execution with \mathcal{A} . That is, the simulator knows all the messages \mathcal{A} sends to the honest parties and hence can also simulate the communication among honest parties. This includes simulating Functionalities 5.4.1, 5.5.1. Note that the simulation may not terminate.

3. If there exists some honest party P_{j^*} that terminates, then it implies that P_{j^*} terminates in the simulation of Functionality 5.4.1. By Claim 5.4.6 we have that all the honest parties eventually terminate.
4. The simulator interpolates the dealer's polynomials from the shares of the lexicographically first $t + 1$ simulated honest parties. It is guaranteed that the polynomials have degree $t + t/2$ in x and t in y .
5. Moreover, since P_{j^*} terminated in the protocol, it also implies that the simulation of Functionality 5.5.1 terminated with the output OK.
6. The simulator sends to the functionality the interpolated polynomials ensuring that all the honest parties will eventually receive the output. When the adversary delivers OK to a party P_j in the simulated protocol, the simulator delivers the output of Functionality 5.7.1 to P_j in the ideal world.

It is easy to see that since the honest parties do not have inputs and the simulator emulates the honest parties as in the real world, the view of the adversary in the ideal and real execution is the same. Moreover, if an honest party P_{j^*} terminates in the real execution, then the same holds true in the simulated execution. The simulator extracts the input polynomials of the dealer from the view of these simulated honest parties, and sends it to the functionality ensuring that the output of honest parties is identical in the ideal model. \square

5.7.1 Batching for Linear overhead per triple

We note that the overall communication of one instance of verifiable triple sharing protocol is $\mathcal{O}(n^3 \log n)$ over point-to-point channels and $\mathcal{O}(n^2 \log n)$ using broadcast. Using the broadcast of [38], the total cost turns out to be $\mathcal{O}(n^4 \log n)$ for sharing $\mathcal{O}(n^2)$ triples. We make the cost linear per triple by simply batching n instances of the triple sharing protocol under the same dealer. Since all the instances have the same dealer, the broadcasts communication can be common for all. For instance, P_i can send a single broadcast of $\text{Good}(i, j)$ after checking consistency with P_j in all the instances of AVSS and triple sharing. Similarly, the dealer can run the Star algorithm just once for all the AVSSs and broadcast one common Star. Likewise, Algorithm 5.5.10 also is run for all the trivariate sharings together and the output is broadcast once for all.

This batching allows us to keep the broadcast communication the same as before i.e. $\mathcal{O}(n^4 \log n)$. The point-to-point communication increases by a factor of n and now becomes $\mathcal{O}(n^4 \log n)$. However, we are now able to share n^3 triplets, and thus achieve a linear overhead.

5.8 Linear Perfectly Secure AMPC

In this section, we give the details of the building blocks required for the complete MPC protocol such as reconstruction of degree- t polynomials, and Beaver triple generation. We conclude with the complete MPC protocol which relies on these building blocks, the packed VSS (Section 5.4) and the verifiable triple secret sharing (Section 5.7) protocol.

5.8.1 Secret Reconstruction

At the termination of our packed VSS, the secrets are available in Shamir-shared format. We discuss how such sharing can be reconstructed efficiently. We use two standard ways of reconstruction:

Private reconstruction. Here, we describe the private reconstruction of a degree- t shared secret to a specified party, say P^* . For this, all the parties disclose their shares to P^* , who tries to recover the secret as follows. P^* waits for $2t + 1$ shares, all of which lie on the same degree- t polynomial. This requires P^* to apply the Reed Solomon (RS) error correction repeatedly in an “online” manner, also known as online error correction (OEC) [40]. If P^* obtains such a polynomial, it is guaranteed to be the correct degree- t polynomial since it agrees with the shares of at least $t + 1$ honest parties. The protocol Π_{Rec} appears below.

Protocol 5.8.1: Private Reconstruction Protocol – Π_{Rec}

Common input: The description of a field \mathbb{F} , n non-zero distinct elements $1, \dots, n$, identity of a party P^* .

Input: Parties hold the univariate degree- t sharing $\langle v \rangle$.

1. Each P_i sends its share $\langle v \rangle_i$ to P^* and terminates.
 2. For $r = 0, \dots, t$:
 - (a) Upon receiving the first $2t + 1 + r$ values, P^* looks for a codeword of a polynomial of degree- t with a distance of at most r from the values it received. If there is no such unique codeword, P^* proceeds to the next iteration. Otherwise, it sets $p_r(x)$ to be the unique RS reconstruction.
 - (b) If $p_r(i) = \langle v \rangle_i$ holds for at least $2t + 1$ parties whose shares were considered during RS reconstruction, then P^* outputs $p_r(0)$ and terminates. Otherwise, it proceeds to the next iteration.
-

Lemma 5.8.2. Protocol 5.8.1, Π_{Rec} , has a communication complexity of $\mathcal{O}(n \log n)$ bits over point-to-point channels and no broadcast for privately reconstructing a value (i.e., $\mathcal{O}(\log n)$ bits) in constant runtime.

Batched public reconstruction. Naïvely, reconstructing $t + 1$ secrets that are Shamir-shared requires $(t + 1)n$ private reconstructions (via Π_{Rec}), resulting in $\mathcal{O}(n^3 \log n)$ communication¹. On the other hand the batch reconstruction protocol, first presented in [57], allows parties to robustly reconstruct $t + 1$ Shamir-shared values at a cost of communicating $\mathcal{O}(n^2 \log n)$ bits, ensuring an amortized cost of $\mathcal{O}(n \log n)$ bits per reconstruction.

In particular, given $\langle v_0 \rangle, \dots, \langle v_t \rangle$, parties translate them to n sharings *non-interactively*, say $\langle v'_1 \rangle, \dots, \langle v'_n \rangle$, using a linear error correcting code, such as Reed-Solomon code which tolerates up to t errors. To be specific, (v'_1, \dots, v'_n) can be thought of as n points on a t -degree polynomial $p(x) = \sum_{i=0}^t v_i x^i$. Following this, of the n sharings, one sharing $\langle v'_i \rangle$ is reconstructed towards each party P_i via private reconstruction protocol Π_{Rec} who obtains v'_i . At this stage, the parties essentially hold $\langle v_0 \rangle$. Therefore, n instances of private reconstruction enables every party to recover $p(x)$, the polynomial used to share v_0 , whose coefficients are the desired output. This requires a total communication of $\mathcal{O}(n^2 \log n)$ bits. The protocol Π_{bPubRec} appears below for completeness.

Protocol 5.8.3: Batched Public Reconstruction Protocol – Π_{bPubRec}

Common input: The description of a field \mathbb{F} , n non-zero distinct elements $1, \dots, n$.

Input: Parties hold the univariate degree- t sharings $\langle v_0 \rangle, \dots, \langle v_t \rangle$.

1. Let $p(x) = v_0 + v_1 x + v_2 x^2 + \dots + v_t x^t$.
 2. For each P_i , parties locally compute $\langle v'_i \rangle = \langle p(i) \rangle = \langle v_0 \rangle + \langle v_1 \rangle \cdot i + \langle v_2 \rangle \cdot i^2 + \dots + \langle v_t \rangle \cdot i^t$.
 3. For each party P_i , parties invoke Π_{Rec} with $\langle v'_i \rangle$ as input to enable P_i to privately reconstruct $v'_i = p(i)$. Note that parties now hold $\langle p(0) \rangle$.
 4. For each party P_i , parties invoke Π_{Rec} with $\langle p(0) \rangle$ as input to enable P_i to privately reconstruct the polynomial $p(x)$. Upon reconstructing, each P_i outputs the $t + 1$ coefficients v_0, v_1, \dots, v_t of $p(x)$.
-

Lemma 5.8.4. Protocol 5.8.3, Π_{bPubRec} , has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and no broadcast for publicly reconstructing $\mathcal{O}(n)$ values (i.e., $\mathcal{O}(n \log n)$ bits) simultaneously and has constant runtime.

¹Alternatively, $(t + 1)n$ elements of broadcasts. With each party broadcasting $(t + 1)$ elements, this results in a cost of $\mathcal{O}(n^4 \log n)$ bits of communication.

5.8.2 The complete MPC protocol

We now describe our MPC protocol using the packed VSS (Section 5.4), the verifiable triple sharing and the building blocks which are taken from [50]. Our MPC protocol relies on Beaver’s circuit randomization trick [23] and has two phases: (i) Preparing the Beaver triples and input sharing, and (ii) Evaluation using the batched Beaver multiplication.

5.8.2.1 Preparing the Beaver Triples and Input Sharing

This phase is further divided into three tasks. First, using the verifiable triple sharing protocol, each party acting as a dealer is made to Shamir-share the required number of triples (a, b, c) such that $c = ab$ holds. Each party also uses the packed VSS described in Section 5.4 to share its inputs. Note however that due to the asynchronous nature of the network, parties cannot afford to wait for the triple sharing and input sharing instances of all the parties to terminate. Doing so might result in an endless wait since the corrupt parties might remain silent and not initiate an instance of sharing. Given this, the parties are required to agree on a common set, say *Core* of (at least) $n - t$ dealers whose triple sharing as well as input sharing instances will eventually terminate for all the parties. This forms the second task, wherein parties execute an instance of ACS [40] (Functionality 5.3.1) to agree on a set of parties whose shared triples and inputs will be considered for subsequent computation. For the parties outside this set, a default sharing of 0 is considered as the input. Finally, once the common set *Core* is decided upon and the triple sharing instances of all the dealers in *Core* terminate, parties execute the triple extraction protocol which uses the triples shared by these parties and gives as output random triples, not known to any party.

Verifiable Triple Sharing and Input Sharing. In this phase, each party shares verified multiplication triples, which will be used in the subsequent phases for extracting random triples unknown to any party. The exact number of triples to be shared by each party depends on the size of the circuit to be computed. We provide the detailed analysis of this in the proof of Theorem 5.8.11. Towards that end, each party invokes the triple sharing functionality (Functionality 5.7.1) in parallel to share the required number of triples. Each party invokes the Functionality 5.4.1 in parallel to share its inputs. However, to ensure termination while accounting for the asynchronous network, parties cannot afford to wait for the triple sharing and input sharing instances of all the parties to terminate. Moreover, waiting for at least $n - t$ parties’ instances to terminate before proceeding to triple extraction does not offer a solution. Honest parties might terminate instances in different sequence leading to inconsistency in the subsequent phase. This issue is tackled by the second phase described below.

Agreement on a Core Set (ACS). Here, parties execute an instance of ACS [40] (Functionality 5.3.1) to agree on a common set of at least $n - t$ parties whose triple sharing and input sharing instances are guaranteed to terminate eventually for all the parties. Having agreed on the set, parties proceed to the final task which consumes the triples shared by each party in this set for extracting random triples unknown to any party.

Triple Extraction. Our last component is a triple extraction protocol that consumes one (verified) multiplication triple, say $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$, shared by each party $P_i \in \text{Core}$ in the prior stage and extracts $h + 1 - t$ random triples not known to any party, where $h = \lfloor \frac{|\text{Core}| - 1}{2} \rfloor$. For simplicity, let $m = |\text{Core}|$ and without loss of generality, we assume $\text{Core} = \{P_1, \dots, P_m\}$. The protocol incurs a cost of $\mathcal{O}(n^2)$ point to point communication and at a high level, proceeds as follows. First, the parties “transform” the m random shared triples $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$ for each $i \in [m]$ into m correlated triples $(\langle x_i \rangle, \langle y_i \rangle, \langle z_i \rangle)$ for every $i \in [m]$ such that the values $\{x_i, y_i, z_i\}_{i \in [m]}$ lie on the polynomials $X(\cdot), Y(\cdot)$ and $Z(\cdot)$ of degree h, h and $2h$ respectively where $X(\cdot) \cdot Y(\cdot) = Z(\cdot)$. Specifically, for each $i \in [m]$, it holds that $X(i) = x_i, Y(i) = y_i$ and $Z(i) = z_i$ where $1, \dots, m$ are publicly known distinct elements from \mathbb{F} . Furthermore, the transformation ensures that the adversary knows $\{x_i, y_i, z_i\}$ only if P_i is corrupt. This implies that the adversary may know (at most) t points on each of the polynomials $X(\cdot), Y(\cdot)$ and $Z(\cdot)$ of degree h, h and $2h$ respectively, thus guaranteeing a degree of freedom of $h + 1 - t$ in $X(\cdot), Y(\cdot)$ (and thus $Z(\cdot)$). Parties thus output the shared evaluation of these polynomials at $h + 1 - t$ publicly known points $\beta_1, \dots, \beta_{h+1-t}$ as the extracted shared multiplication triples.

The transformation itself works as follows. The parties simply set $x_i = a_i, y_i = b_i, z_i = c_i$ for $i \in \{1, \dots, h + 1\}$. Next, $\langle x_i \rangle$ and $\langle y_i \rangle$ for every $i \in \{h + 2, \dots, m\}$ can be computed *non-interactively* by taking linear combination of $\{x_i, y_i\}_{i \in [h+1]}$. Following this, $\langle z_i \rangle$ for every $i \in \{h + 2, \dots, m\}$ is computed using Beaver’s trick where the inputs are $\langle x_i \rangle$ and $\langle y_i \rangle$ and the triple $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$. Clearly, if P_i is corrupt then x_i, y_i, z_i is known to the adversary as claimed. To conclude, we note that triple extraction reduces to running a batch of $\mathcal{O}(h)$ Beaver multiplications which requires $\mathcal{O}((nh + n^2) \log n)$ bits communication using Π_{bPubRec} . The formal description appears in Protocol 5.8.5.

Protocol 5.8.5: Triple Extraction – $\Pi_{\text{tripleExt}}$

Common input: The description of a field \mathbb{F} , a set $\text{Core} \subseteq \mathcal{P}$ such that $m = |\text{Core}|$, $m = 2h + 1$ non-zero distinct elements $1, \dots, m$ and $h + 1 - t$ non-zero distinct elements $\beta_1, \dots, \beta_{h+1-t}$. Without loss of generality, assume $\text{Core} = \{P_1, \dots, P_m\}$.

Input: Parties hold the degree- t shared triples $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$ for every $i \in [m]$ such that (a_i, b_i, c_i) is known to party P_i .

1. For each $i \in [h + 1]$, parties locally set $\langle x_i \rangle = \langle a_i \rangle$, $\langle y_i \rangle = \langle b_i \rangle$ and $\langle z_i \rangle = \langle c_i \rangle$.
 2. Let $X(\cdot)$ and $Y(\cdot)$ be the degree- h polynomials defined by the points $\{x_i\}_{i \in [h+1]}$ and $\{y_i\}_{i \in [h+1]}$ respectively such that $X(i) = x_i$ and $Y(i) = y_i$ for all $i \in [h + 1]$.
 3. For each $i \in \{h + 2, \dots, m\}$, parties locally compute $\langle x_i \rangle = \langle X(i) \rangle$ and $\langle y_i \rangle = \langle Y(i) \rangle$.
 4. Parties invoke Π_{bBeaver} with $\{\langle x_i \rangle, \langle y_i \rangle, \langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle\}_{i \in \{h+2, \dots, m\}}$ and obtain $\{\langle z_i \rangle\}_{i \in \{h+2, \dots, m\}}$ where $z_i = x_i y_i$ for every $i \in \{h + 2, \dots, m\}$.
 5. Let $Z(\cdot)$ be the degree- $2h$ polynomial defined by the points $\{z_i\}_{i \in [m]}$ such that $Z(i) = z_i$ for all $i \in [m]$.
 6. Parties locally compute $\langle \mathbf{a}_i \rangle = \langle X(\beta_i) \rangle$, $\langle \mathbf{b}_i \rangle = \langle Y(\beta_i) \rangle$ and $\langle \mathbf{c}_i \rangle = \langle Z(\beta_i) \rangle$ for every $i \in [h + 1 - t]$.
-

Lemma 5.8.6. *Protocol 5.8.5, $\Pi_{\text{tripleExt}}$ has a communication complexity of $\mathcal{O}((nh + n^2) \log n)$ bits over point-to-point channels and no broadcast for sharing $h + 1 - t$ random multiplication triples in constant runtime.*

5.8.2.2 Batched Beaver Multiplication

This corresponds to the second phase of our MPC protocol, which uses the degree- t shared multiplication triples computed in the prior phase to evaluate the multiplication gates in the circuit via Beaver multiplication in a batched manner. This protocol relies on the well known technique of Beaver’s circuit randomization [23], which, given a pre-computed t -shared random and private multiplication triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, reduces the computation of $\langle xy \rangle$ from $\langle x \rangle$ and $\langle y \rangle$ to two public reconstructions. Towards this, parties first locally compute $\langle d \rangle = \langle x \rangle - \langle a \rangle$ and $\langle e \rangle = \langle y \rangle - \langle b \rangle$, followed by public reconstruction of d and e . Since $z = xy = ((x - a) + a)((y - b) + b) = (d + a)(e + b) = de + db + ea + ab$, parties can locally compute $\langle z \rangle = \langle xy \rangle$ using the shared multiplication triple and the publicly reconstructed values d and e . Specifically, parties locally compute $\langle xy \rangle = de + d\langle b \rangle + e\langle a \rangle + \langle c \rangle$.

To leverage the efficiency benefits offered by the batch public reconstruction protocol, the protocol handles a batch of l multiplications together, each requiring 2 reconstructions. The $2l$ public reconstructions are thus batched together in groups of $t + 1$ to invoke Π_{bPubRec} and ensure an amortized communication complexity of $\mathcal{O}(n \log n)$ bits per reconstruction. The resultant communication complexity of Π_{bBeaver} for handling l multiplications is $\mathcal{O}((n^2 + nl) \log n)$. The formal description appears in Protocol 5.8.7.

Protocol 5.8.7: Batched Beaver Multiplication – Π_{bBeaver}

Input: Parties hold l degree- t shared triples $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$ for every $i \in [l]$ and l degree- t shared pairs of values $(\langle x_i \rangle, \langle y_i \rangle)$ to be multiplied.

1. For each $i \in [l]$, parties locally compute $\langle d_i \rangle = \langle x_i \rangle - \langle a_i \rangle$ and $\langle e_i \rangle = \langle y_i \rangle - \langle b_i \rangle$.
 2. Let $2l = k(t+1)$. Parties execute k parallel instances of Π_{bPubRec} and publicly reconstruct $\{d_i, e_i\}$ for every $i \in [l]$.
 3. For each $i \in [l]$, parties locally compute $\langle z_i \rangle = \langle x_i y_i \rangle = d_i e_i + d_i \langle b_i \rangle + e_i \langle a_i \rangle + \langle c_i \rangle$.
-

Lemma 5.8.8. *Protocol 5.8.7, Π_{bBeaver} , has a communication complexity of $\mathcal{O}((ln + n^2) \log n)$ bits over point-to-point channels and no broadcast for the multiplication of l pairs of shared values and has constant runtime.*

5.8.3 The MPC Protocol

We first describe the MPC functionality, followed by the complete protocol using the building blocks described above. We subsequently provide the proof of security and the communication complexity analysis of the protocol.

Functionality 5.8.9: AMPC – $\mathcal{F}_{\text{AMPC}}$

The functionality is parameterized by a set of corrupted parties $I \subseteq [n]$. Initialize the sets $S, H, I' = \emptyset$. Initialize $x_i = 0$ for every $i \in I$.

Input: Each P_i holds input $x_i \in \mathbb{F} \cup \{\perp\}$.

Common Input: An n -party function $f(x_1, \dots, x_n)$.

1. Upon receiving (Input, j, x_j) from an honest party P_j , if $j \notin H$ then add j to S .
 2. Receive from the adversary, the sets $H \subset [n] \setminus I$ and $I' \subseteq I$ such that $|H| \leq |I'| \leq t$. Also receive a set of inputs $\{(\text{Input}, i, x_i)\}_{i \in I'}$.
 3. If $|S| < n - t$, then for each P_j with $j \in H$, the functionality sets $x_j = 0$ and updates $S = S \setminus H$.
 4. If $|S \cup I'| \geq n - t$, then compute $(y_1, \dots, y_n) = f(x_1, \dots, x_n)$ and send y_i to P_i for every $i \in [n]$ and terminate.
-

Protocol 5.8.10: AMPC – Π_{AMPC}

Common input: The description of a circuit, the field \mathbb{F} , n non-zero distinct elements $1, \dots, n$ and a parameter h where $n - t = 2h + 1$. Let $m = \lceil \frac{C}{h+1-t} \rceil$.

Input: Parties hold their inputs (belonging to $\mathbb{F} \cup \{\perp\}$) to the circuit.

(Beaver triple generation and Input sharing:)

1. **(Beaver Triple generation with a dealer)** Each P_i chooses m random multiplication triples and executes $\lceil \frac{m}{(t/2+1)^2} \rceil$ instances of Protocol 5.7.2 (Section 5.7) in a batched manner each with $(t/2 + 1)^2$ triples.
2. **(Input sharing)** Each party P_i holding k_i inputs to the circuit executes the VSS protocol (Functionality 5.4.1) in a batched manner, packing $\lceil \frac{k_i}{t/2+1} \rceil$ inputs in one instance.
3. **(ACS Execution)** Parties invoke ACS protocol (Functionality 5.3.1) to agree on a set Core of at least $n - t$ parties whose instances of triple sharing and input sharing will terminate eventually all the honest parties. Let $(\langle a_i^j \rangle, \langle b_i^j \rangle, \langle c_i^j \rangle)$ for $j \in [m]$ denote the triples shared by $P_i \in \text{Core}$. The input sharing for the parties outside Core is take as default sharing of 0.
4. **(Beaver Triple Extraction)** Parties execute m instances of the triple extraction protocol, $\Pi_{\text{tripleExt}}$ (Protocol 5.8.5), with Core as the common input and additionally $(\langle a_i^j \rangle, \langle b_i^j \rangle, \langle c_i^j \rangle)$ for every $P_i \in \text{Core}$ as the input for the j^{th} instance. Let $(\langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle)$ for $i \in [C]$ denote the random multiplication triples generated.

(Circuit computation:)

1. **(Linear Gates)** Parties locally apply the linear operation on their respective shares of the inputs.
2. **(Multiplication Gates)** Let $(\langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle)$ be the multiplication triple associated with the i^{th} multiplication gate with shared inputs $(\langle x_i \rangle, \langle y_i \rangle)$. Parties invoke the batched Beaver protocol, Π_{bBeaver} (Protocol 5.8.7), with $\{\langle x_i \rangle, \langle y_i \rangle, \langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle\}$ for all gates i at the same layer of the circuit and obtain the corresponding $\langle z_i \rangle$ as the output sharing for every gate i .
3. **(Output)** For each output gate j with the associated sharing $\langle v_j \rangle$, parties execute private reconstruction protocol, Π_{Rec} (Protocol 5.8.1), towards every party P_i who is supposed to receive the output v_j .

Theorem 5.8.11. *Let $n \geq 4t + 1$. Protocol 5.8.10 securely computes Functionality 5.8.9 in the Functionality (5.4.1, 5.7.1, 5.3.1)-hybrid in the presence of a malicious adversary controlling at*

most t parties. It has a communication complexity of $\mathcal{O}((Cn + Dn^2 + n^5) \log n)$ bits over point-to-point channels and $\mathcal{O}(n^3 \log n)$ bits of broadcast for evaluating a circuit with C gates and depth D . Each party broadcasts at most $\mathcal{O}(n^2 \log n)$ bits.

Proof. The circuit evaluation requires C random shared multiplication triples. We analyze the cost of the two phases of the MPC protocol separately.

Beaver triple generation and Input sharing. Note that one instance of the triple extraction protocol (Protocol 5.8.5) extracts $h + 1 - t$ random triples simultaneously, where $h = \lceil \frac{|\text{Core}| - 1}{2} \rceil$. Given that Core is of size at least $n - t$, we have that $h + 1 - t = \mathcal{O}(n)$. Hence, we require $\mathcal{O}(C/n)$ instances of the triple extraction, which incurs a cost of $\mathcal{O}((Cn + n^2) \log n)$ bits of communication. Further, each instance of the triple extraction protocol consumes one verified triple shared by each party. Hence, each party is required to share $\mathcal{O}(C/n)$ multiplication triples. Since our verifiable triple sharing protocol packs $\mathcal{O}(n^2)$ triples in one instance, it requires each party to execute $\mathcal{O}(C/n^3)$ instances of the protocol in a batched manner, which results in a cost of $\mathcal{O}((C + n^3) \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits of broadcast per dealer. That is, the total cost incurred is $\mathcal{O}((Cn + n^5) \log n)$ using the protocol of [37] to instantiate broadcast. Moreover, the ACS protocol has a cost of $\mathcal{O}(n^5 \log n)$ (as discussed in Section 5.2.3), resulting in a communication complexity of $\mathcal{O}((Cn + n^5) \log n)$ for this phase.

Circuit evaluation. Here, parties evaluate batches of multiplication gates at the same level in the circuit by invoking the batched Beaver multiplication protocol (Protocol 5.8.7). Given C_i is the number of gates per level of the circuit, this stage incurs a cost of $\mathcal{O}(C_i \cdot n \log n + n^2 \log n)$ bits communication over point-to-point channels. Thus we have that this phase requires a total of $\sum_{i=1}^D \mathcal{O}(C_i \cdot n \log n + n^2 \log n) = \mathcal{O}(Cn \log n + Dn^2 \log n)$ bits communication over point-to-point channels.

We now show the simulation. The simulator \mathcal{S} is as follows:

1. Upon activation, invoke the adversary \mathcal{A} .
2. *Beaver triple generation and Input Sharing*
 - (a) On behalf of each honest party P_j , the simulator chooses m random multiplication triples and simulates the Functionality 5.7.1 for the case of an honest dealer by choosing the appropriate bivariate polynomials.
 - (b) On behalf of each honest party P_j holding an input to the circuit, the simulator chooses a value uniformly at random and simulates the Functionality 5.4.1 for the case of an honest dealer.

- (c) For every $i \in I$, simulate Functionality 5.7.1 and 5.4.1 as a functionality would run it. If the simulation of Functionality 5.4.1 terminates, then from the shares of the honest parties, the simulator reconstructs the input x_i .
- (d) Simulate the Functionality 5.3.1: When the adversary delivers the shares to an honest party P_j in the simulation of instance of Functionality 5.7.1 and Functionality 5.4.1 (if any) corresponding to some party P_k , simulate P_j sending (record, j, k) . Listen to the record messages sent by the adversary. Simulate the functionality sending a set S upon receiving the command $\text{receiveS}()$ from the adversary. When $S \neq \perp$, let $\text{Core} = S$.
- (e) The simulator has the shares of honest parties corresponding to the triples shared by each party in S when $S \neq \perp$, and hence can run Protocol 5.8.5 as honest parties would. Specifically, it can simulate all the communication from honest parties including that among honest parties as in the protocol.

3. Circuit evaluation

- (a) The simulator holds shares of the honest parties corresponding to the inputs of each party in Core , as well as the shares corresponding to the default input 0 of the remaining parties.
- (b) On behalf of each honest party, the simulator computes the linear operations on the shares of honest parties as in the protocol.
- (c) For each multiplication gate, the simulator holds the shares of honest parties and can run Protocol 5.8.7 as honest parties would run it.
- (d) The simulator constructs a set H of all the simulated honest parties not in Core . It also constructs a set $I' = \text{Core} \cap I$. It invokes Functionality 5.8.9 with the sets H, I' and the extracted inputs $\{(\text{Input}, i, x_i)\}_{i \in I'}$.
- (e) It receives from the functionality the outputs y_i for every $i \in I$.
- (f) The simulator computes the shares of the honest parties to be used in the simulation of Protocol 5.8.1 using the output y_i and the shares of the corrupt parties (which can be computed from the view of the simulated honest parties). The simulator runs Protocol 5.8.1 as honest parties would using these shares.
- (g) When the adversary allows successful reconstruction of the output towards an honest party P_j in the simulated protocol, the simulator delivers the output of Functionality 5.8.9 in the ideal protocol.

We now show that the view of the adversary is indistinguishable in both the executions. We do this using a sequence of hybrids.

Hybrid₀: Execution of Π_{AMPC} in the real world.

Hybrid₁: In this hybrid, the simulator simulates the execution of the output gates as described. The only change is that the simulator computes the inputs of the corrupt parties in Core , and the sets H and I' and invokes the Functionality 5.8.9 with these inputs. Note that for the output y_i towards a corrupt P_i , the shares of honest parties in the reconstruction are completely determined by the shares of the corrupt parties and the output y_i . Hence, the shares computed by \mathcal{S} are identical to the real shares of the honest parties. The distributions of **Hybrid₀** and **Hybrid₁** are identical.

Hybrid₂: Here, the simulator simulates the execution of the addition and multiplication gates. Since \mathcal{S} knows the shares held by honest parties, it can simulate the addition and multiplication gates as honest parties would in the real protocol. The distributions of **Hybrid₁** and **Hybrid₂** are identical.

Hybrid₃: In this hybrid, \mathcal{S} simulates the execution of triple extraction as described. Since the shares of honest parties are held by the simulator, the two distributions are identical.

Hybrid₄: In this hybrid, the simulator simulates the Functionality 5.3.1. By the security of ACS, we have that the two distributions are indistinguishable.

Hybrid₅: In this hybrid, \mathcal{S} simulates the invocations of Functionality 5.7.1 and Functionality 5.4.1 by choosing triples and inputs respectively uniformly at random on behalf of the honest parties. The two hybrids differ only in the manner that \mathcal{S} uses the random triples and inputs of honest parties in **Hybrid₄**, whereas it samples random values in **Hybrid₅**. The shares of the corrupt parties are distributed uniformly at random in both the hybrids. Due to Theorem 5.7.3 and Theorem 5.4.4, we have that the distributions in **Hybrid₄** and **Hybrid₅** are indistinguishable.

Note that **Hybrid₅** is the execution between \mathcal{S} and \mathcal{A} in the ideal world. Therefore, we conclude that the distributions in **Hybrid₀** and **Hybrid₅** corresponding to the executions in the real and ideal worlds respectively are indistinguishable.

Note that in the simulated execution, the circuit evaluation is performed considering the inputs shared by parties in Core and with default inputs for the remaining parties. The simulator invokes the functionality in the ideal world with the sets H and I' such that the inputs of the same set of parties in Core are considered during computation of the function, ensuring that the output of the honest parties is the same in both the executions. \square

Chapter 6

Perfectly-secure Network-agnostic MPC with Optimal Resiliency

In this chapter, we discuss the feasibility of the perfectly-secure network-agnostic MPC. Towards this end, we prove the lower bound on the corruption threshold necessary to construct such protocols. Further, to match this bound and show sufficiency, we provide an MPC protocol for the optimal corruption threshold. For this, we give improved constructions for weak secret sharing, verifiable secret sharing and verifiable triple sharing with optimal corruption threshold.

6.1 Introduction

The feasibility of perfectly-secure MPC for synchronous and asynchronous settings has been established decades ago. Specifically, [94, 29] show that perfectly-secure MPC in the synchronous setting tolerating t_s active corruption is possible if and only $t_s < n/3$. Similarly, it is known that perfect security in the asynchronous setting can be achieved as long as the number of corrupt parties is $t_a < n/4$ [30, 31, 5]. However, the feasibility question of perfectly-secure network-agnostic MPC, where the parties are unaware of the network type, has remained open although there have been conjectures in prior works. In particular, [13] conjectures the necessity of $n > 3t_s + t_a$ and shows sufficiency of such a protocol tolerating t_s active corruptions when the network is synchronous and t_a active corruptions when the network is asynchronous. Surprisingly, it is not known if the bound is tight.

Our Main Result

In this work, we completely settle the feasibility of perfectly-secure network-agnostic MPC. We prove the following theorem.

Theorem 6.1.1 (Main Result). *There exists a perfectly-secure, network-agnostic MPC protocol that is secure against an adversary corrupting up to t_s parties in a synchronous network and up to t_a parties in the asynchronous network if and only if $n > 2 \cdot \max(t_s, t_a) + \max(2t_a, t_s)$.*

Note that when $t_s \leq t_a$, our result gives a bound of $n > 4t_a$, which is the known lower bound for asynchronous MPC protocols. Also, as observed by the prior works, any known MPC protocol designed for the asynchronous network with this bound will be trivially secure in the synchronous network, thus serving as the network-agnostic protocol. For the other case, when $t_s > t_a$, we further have two cases to consider. First, when $2t_a \geq t_s$, we obtain a bound of $n > 2t_s + 2t_a$. Whereas when $2t_a < t_s$, we have that $n > 3t_s$ is necessary and sufficient. Thus, we show that the threshold $n > 3t_s + t_a$ used in the prior works on perfectly-secure network-agnostic protocols is not tight for this setting.

Main Technical Result

Our main result is obtained via two key components– the necessity and the sufficiency.

Theorem 6.1.2 (Necessity). *For any n , if $2 \cdot \max(t_s, t_a) + \max(2t_a, t_s) \geq n$, then there is no perfectly-secure n -party MPC protocol that is secure against an adversary corrupting t_s parties in the synchronous network and t_a parties in the asynchronous network.*

Due to reasons discussed earlier, in the rest of the discussion, we focus our attention on the case when $t_a < t_s$. Within this case, when $2t_a \leq t_s$, the impossibility of $n \leq 3t_s$ is inherited from the impossibility of the synchronous setting. So, the most interesting case is that of $n \leq 2t_s + 2t_a$ when $2t_a > t_s$, which we prove. For simplicity, we assume $n = 2t_s + 2t_a$ and show that no network-agnostic perfectly-secure protocol with $2t_s + 2t_a$ parties can compute a specific function f (described below) securely when the network is asynchronous. For this, we assume the existence of an n -party network-agnostic protocol with $n = 2t_s + 2t_a$ and arrive at a contradiction as follows. We first reduce the n party network-agnostic protocol to a 4 party protocol with parties P_1, P_2, P_3, P_4 such that P_1, P_2 emulate disjoint sets of t_s parties each, and each of P_3, P_4 emulate t_a parties in the underlying protocol. Next, we identify a function f as follows

$$f(x_1, x_2, \perp, \perp) \rightarrow (x_1 \wedge x_2, x_1 \wedge x_2, \perp, \perp)$$

Since the n party network-agnostic protocol is secure against an adversary corrupting t_s parties in the synchronous network and t_a parties in the asynchronous network, we have that the 4 party protocol should be secure if P_1 or P_2 is corrupt when run in a synchronous network, or one of P_3, P_4 is corrupt in an asynchronous network. We conclude our proof by

demonstrating that it is impossible for the output receiving parties P_1, P_2 to have a unanimous output when the protocol is instantiated in the asynchronous network where either P_3 or P_4 is corrupt.

Our second contribution lies in providing a matching upper bound. In our view, the most technically involved contributions here are the weak secret sharing and verifiable triple sharing protocols. Weak secret sharing is a primitive with the following properties: (i) privacy: after the sharing phase, the adversary cannot learn anything about the secret of an honest dealer; (ii) commitment: the secret is completely determined by the shares of the honest parties after the sharing phase completes, however, all the honest parties may not necessarily have their shares; and (iii) correctness: if the dealer is honest, then at the end of the sharing phase, all the honest parties hold their shares corresponding to the dealer's secret. On the other hand, verifiable triple sharing allows a dealer to share multiplication triples such that all the parties can verify their correctness. In the network-agnostic setting, since the parties are unaware of the network conditions, the protocols must tolerate the worst case corruption threshold. Hence, the protocols typically operate with the sharing threshold of t_s ($> t_a$). Our construction of both the primitives, weak secret sharing as well as verifiable triple sharing crucially relies on utilizing the additional $t_s - t_a$ degree of freedom which is inherently available when the protocol operating with threshold t_s is instantiated in the asynchronous network where at most t_a parties can be corrupt. Carefully leveraging this degree of freedom while being unaware of the exact network type constitutes our work's primary technical contribution, allowing us to obtain a protocol matching the lower bound. A verifiable secret sharing is built using the weak secret sharing to ensure that all the honest parties have shares even for a corrupt dealer. The verifiable secret sharing then serves as a building block in the triple secret sharing which acts as the primary tool for generating random multiplication triples, the main ingredient for MPC.

Theorem 6.1.3 (Sufficiency). *Let n, t_s, t_a be such that $n > 2t_s + \max(2t_a, t_s)$. There exists a perfectly-secure, network-agnostic MPC protocol for any function secure against an adversary that can corrupt up to t_s parties in the synchronous network and up to t_a parties in the asynchronous network.*

Our weak secret sharing relies on finding a clique of size $n - t_s$ and hence requires exponential time. As discussed in Section 6.2, our weak secret sharing deals with a lot of challenges, despite using a clique finding algorithm. Achieving polynomial time protocols in the optimal threshold is an interesting open problem.

6.1.1 Related Work

We review some other related works below. Network-agnostic computation has been considered in other settings such as the general adversarial structure [14, 12], statistical security [12], computational security [34, 36, 63]. Further, it has been studied for state machine replication [35], secure message transmission [62], and consensus [13]. Importantly, [13] gives network-agnostic protocols for consensus and broadcast with perfect security and a threshold of $t_s, t_a < n/3$ which is known to be optimal for both the synchronous and asynchronous networks.

6.2 Technical Overview

In this section, we provide a technical overview of our work. We first describe the weak and verifiable secret sharing schemes in Section 6.2.1. Verifiable secret sharing protocol allows a designated party, the dealer, to perform a degree- t_s Shamir-sharing (often, abbreviated as t_s -sharing) of its secrets. It is built on top of weak secret sharing via similar techniques used in earlier works [98, 97, 91]. In our MPC protocol, each party shares multiplication triples using the above protocol. Following this, parties must verify the correctness of these degree- t_s shared triples. This is captured by our verifiable triple sharing protocol, which requires additional techniques that are described in Section 6.2.2. Finally, in Section 6.2.3, we conclude with the high-level ideas of how these primitives are utilized to construct the network-agnostic MPC protocol.

6.2.1 Weak and Verifiable Secret Sharing

We start with an approach similar to the prior network-agnostic work of [13] and construct a primitive which we refer to as weak secret sharing (WSS) which proceeds in two phases, *sharing* and *reconstruction*. This primitive is weaker than verifiable secret sharing (VSS) in terms of the guarantees it offers and allows a dealer to share a secret with the following properties:

- **Privacy:** When the dealer is honest, the adversary cannot learn any information regarding the dealer’s secret at the end of the sharing phase.
- **Commitment:** If the dealer is corrupt, at the end of the sharing phase, either no honest party holds its share or a subset of the honest parties hold their shares such that these shares completely determine the dealer’s secret. Moreover, all the parties which hold a share are ensured to have shares corresponding to a common secret.
- **Correctness:** If the dealer is honest, all the honest parties hold shares consistent with

the dealer's secret at the end of the sharing phase.

Although [13] provides a protocol for weak secret sharing, which they call weak polynomial sharing, they assume a threshold of $n > 3t_s + t_a$. We discuss the high level approach of [13], which follows from previous work in this setting [36] and the challenges to extend it to the optimal-resiliency setting. To construct a network-agnostic weak secret sharing protocol, the idea is to run a protocol designed for weak secret sharing in a synchronous network followed by that for an asynchronous network with some intermediate steps to ensure correctness. In more detail, the protocol design relies on observing the properties guaranteed by the synchronous protocol, and either deciding on an output or deciding to run the asynchronous protocol subsequently. Typically, the sharing occurs via a bivariate polynomial, where the dealer sends a univariate polynomial as a share to each party. This is followed by parties checking the pairwise consistency of their univariate polynomial by exchanging one point with each party and broadcasting the result of this check. Subsequently, parties ensure that the dealer has indeed committed to a polynomial (and hence a value) by checking the existence of a clique of sufficiently large size. To ensure this in polynomial time, their protocol uses the (n, t) -Star algorithm [40] whose properties are described below and in Section 6.3. We give a high-level relevant description of their protocol below, bypassing the finer details such as specific time steps and wait periods to ensure correctness.

1. **(Sending polynomial shares)** The dealer chooses a symmetric bivariate polynomial $S(x, y)$ with degree t_s in each variable and the constant term embedding its secret. The dealer then sends to each P_i , its share $S(x, i)$.
2. **(Pairwise consistency check)** Let the polynomial received by P_i be $q_i(x)$. Each P_i sends to every P_j a point $q_i(j)$.
3. **(Broadcasting the results of consistency check)** Let q_{ji} be received by P_i from P_j . P_i broadcasts $\text{OK}(i, j)$ if $q_{ji} = q_i(j)$ holds, and $\text{NOK}(i, j, q_i(j))$ otherwise.
4. **(Constructing the consistency graph)** Each party constructs a graph G with vertices as $\{1, \dots, n\}$ such that an edge (i, j) is included in G if and only if $\text{OK}(i, j)$ and $\text{OK}(j, i)$ is received from the broadcast of P_i, P_j respectively.
5. **(Finding (n, t_s) -Star)** The dealer updates its consistency graph as follows:
 - Remove all the edges incident with P_i if $\text{NOK}(i, j, q_{ij})$ was received from P_i such that $q_{ij} \neq S(i, j)$.
 - From the set of vertices, remove those with degree smaller than $n - t_s$. Perform this step iteratively till no more vertices can be eliminated.

Let the graph induced after these modifications be G_D , and the set of vertices be W . Following this, the dealer runs the (n, t_s) -Star algorithm and broadcasts it if found.

6. **(Deciding on (n, t_s) or (n, t_a) -Star)** Parties run a Byzantine agreement protocol to decide on whether an (n, t_s) -Star was found, or whether to proceed and identify an (n, t_a) -Star.
7. **(Finding (n, t_a) -Star)** If the latter is decided, then the dealer runs the (n, t_a) -Star algorithm and broadcasts it if found.
8. **(Computing the Output)** Finally, parties decide on the output based on the outcome of the byzantine agreement and upon validating the dealer's broadcast of the Star¹.

Protocol in the non-optimal threshold setting [13]. As mentioned, the above protocol by Appan et al. [13] crucially relies on the fact that $n > 3t_s + t_a$. Specifically, consider the case of finding an (n, t_s) -Star in the graph G_D with vertex set as W . The output of the Star algorithm is a pair of sets say (C, D) where $C \subseteq D \subseteq W$ such that $|C| \geq n - 2t_s$ and $|D| \geq n - t_s$, and additionally, there exists an edge between each $i \in C$ and every $j \in D$. This implies $|C| \geq t_s + t_a + 1$ and $|D| \geq 2t_s + t_a + 1$. Their protocol guarantees commitment to a polynomial in the synchronous network by ensuring that all the *honest* parties in W are indeed consistent with each other. We say that parties P_i, P_j are consistent if their pairwise consistency check is successful, thus $\text{OK}(i, j)$ and $\text{OK}(j, i)$ are received from their broadcast respectively. Specifically, the protocol is designed with appropriate timeouts which ensures that if any pair of honest parties have a conflict (their pairwise exchanged points do not match) in the synchronous network setting, then this conflict would be conveyed to all the honest parties before they accept the (n, t_s) -Star. Parties accept the dealer's (n, t_s) -Star, that is the sets C, D , if and only if there are no conflicts among the parties in it and $C, D \subseteq W$. This ensures that if a Star is accepted, then all the honest parties included in W (and hence in C, D) are consistent with each other. Thus, there is a unique bivariate polynomial defined by the honest parties.

Now consider the scenario when the network is asynchronous; however, the adversary behaves similarly to the synchronous case till the honest parties accept (n, t_s) -Star. Now, we cannot argue that the honest parties in W are consistent with each other and define a unique bivariate polynomial based on the timeout argument. A pair of honest parties which are in conflict may be included in W solely due to the delay of their NOK messages, which can never occur in the synchronous network. Instead of the timeout guarantees, the argument for the

¹Validating requires checking certain conditions. We mention the conditions relevant to our discussion when required.

asynchronous case relies the threshold of $n > 3t_s + t_a$. Observe that $|C| \geq n - 2t_s$, and we also have that the adversary can corrupt at most t_a parties in the asynchronous network. This ensures that there are at least $|C| - t_a > t_s$ honest parties in the set C , which are consistent with each other (by the properties of the Star algorithm). These parties thus define a unique bivariate polynomial of degree t_s in each variable. Further, this guarantees that all the parties in D also have their shares on this unique polynomial. This follows from the fact that by the properties of Star algorithm, parties in D are bound to be consistent with all the parties in C , which in turn includes at least $t_s + 1$ honest parties defining the polynomial.

Challenges with optimal-threshold. Translating the above protocol to optimal resilience has immediate problems. In particular, consider the latter case described above, where the network is asynchronous and the parties have accepted an (n, t_s) -Star. The condition $|C| - t_a > t_s$ no longer holds. This in turn implies that there is no unique bivariate polynomial defined by the shares of honest parties in C , and consequently parties in D . Thus, we do not get any guarantees from the synchronous protocol when run in the asynchronous network, which are typically required to ensure correctness. This is one of the primary hurdles in constructing our protocol and requires us to introduce new techniques.

Extending to the optimal resilience. Our first crucial observation is that the issue of ensuring the dealer's commitment can be mitigated if we consider an (n, t_a) -Star regardless of the network type. This is because, in this case, the sets C, D are such that $|C| \geq n - 2t_a$ and $|D| \geq n - t_a$. This guarantees us that $|C| - t_a = n - 3t_a > t_s$, and thus the honest parties in C indeed define a unique bivariate polynomial with their shares. However, we cannot expect an (n, t_a) -Star to be found in the synchronous network even when the dealer is honest. Given that $t_s > t_a$, even for an honest dealer, the biggest clique that the consistency graph may have is of size $n - t_s$. Whereas the Star algorithm guarantees an output of (n, t_a) -Star only when the graph contains a bigger clique of size $n - t_a$. Therefore, we start with a clique of size $n - t_s$ and find a way to expand it to a clique of size $n - t_a$ so that we have an (n, t_a) -Star regardless of the network type.

At a high level, our protocol has the following structure. It follows similar to [13] till the broadcasting of the result of consistency check among parties. Following this, the dealer identifies and broadcasts a clique of size $n - t_s$. If the dealer successfully broadcasts this within a designated time, then parties proceed to the clique extension phase. Otherwise, it is guaranteed that the dealer is either corrupt in a synchronous network, or the network is asynchronous. To handle this, parties run an agreement and immediately decide to switch modes and expect the dealer to broadcast a clique of size $n - t_a$. We now discuss the clique

extension phase.

The extension combines the following observations to satisfy our requirements while maintaining privacy in each network condition. First, we observe that in the synchronous network, a pair of honest parties will broadcast the outcome of their pairwise consistency checks within a designated time. Thus, when the dealer is honest, if any pair of parties does not have an edge between them by this time, then it is assured that at least one of these parties is corrupt. Hence, we can publicly reveal the common point these parties hold without breaching privacy. On the other hand, if the network is asynchronous, this claim does not hold true. That is, a pair of parties without an edge may indeed be slow honest parties whose broadcast is delayed. Hence, such a revelation of points leads to the adversary learning more points on the polynomial. However, we observe that the protocol operating with degree (t_s, t_s) bivariate polynomial in the asynchronous network has an additional degree of freedom of $t_s - t_a$. We leverage this freedom to ensure privacy in the asynchronous network. Precisely, the dealer first identifies a clique of the maximum possible size in the consistency graph. We are done if the clique is already of size $n - t_a$. Otherwise, we expect a clique of size at least $n - t_s$. An honest dealer in synchronous network will surely find such a clique consisting of all the honest parties. To extend the clique, the dealer identifies at most $t_s - t_a$ additional parties it wishes to include in the clique as follows.

First, the dealer identifies if any party broadcast an incorrect value during pairwise consistency check or was silent in the consistency check of more than t_s parties. If it finds such parties, it includes them in a set U . Following this, it instructs all the parties to restart the protocol with the polynomials of parties in U now being public. Note that each party added to U by an honest dealer in the synchronous network is guaranteed to be corrupt and will now be forced to behave honestly, thus increasing the clique size by $|U|$. Hence, we have that once U is of size $t_s - t_a$, the dealer will succeed in identifying an $n - t_a$ sized clique consisting of all the honest parties along with the corrupt parties in U . If the dealer finds no such party that can be added to U , then it approaches the clique extension via an alternative technique. Here, the dealer arbitrarily identifies a set of $t_s - t_a - |U|$ parties, say V , outside the clique. Thereafter, it instructs all parties not yet marked consistent with V to broadcast their pairwise points, and similarly, parties in V broadcast their corresponding points. Observe that if all the parties indeed broadcast their correct points within the designated time of the synchronous network, then we have that the clique expands to size $n - t_a$. If not, then the dealer once again is able to identify the parties that are silent or broadcast an incorrect value and add them to U . Following this, similar to the earlier discussion, it instructs parties to restart the protocol. We stress that we limit the number of parties added to U to $t_s - t_a$. While in the

synchronous network, the designated time steps ensure privacy for an honest dealer by only adding corrupt parties to U , privacy is maintained even in the asynchronous network due to the public revelation of at most $t_s - t_a$ polynomials of honest parties. Together with the t_a polynomials of the corrupt parties, the adversary may learn at most t_s univariate polynomial shares on the dealer's (t_s, t_s) -degree bivariate polynomial which still ensures privacy. It is worth noting that the number of reruns may go up to $t_s - t_a$.

We will briefly discuss how each party computes its share in the weak secret sharing protocol after accepting a *fully-consistent* clique of size $n - t_a$, where all the parties in the clique are pair-wise consistent. Since the clique has at least $n - t_a - t_s > t_s + \max(t_a, t_s - t_a)$ honest parties, their shares define a unique bivariate polynomial of degree t_s in both variables. Hence, a party inside the clique can output the univariate polynomial it received from the dealer and used during pairwise consistency check. On the other hand, a party lying outside the clique is required to obtain its polynomial share which is consistent with the honest parties in the clique. For this, the parties in the clique send their pairwise common points to a party outside the clique. Again, we use some crucial observations, as below, to ensure that an honest party outside the clique indeed reconstructs a correct polynomial in all cases except when the network is synchronous and the dealer is corrupt. It is because of this exception our protocol does not qualify to be a verifiable secret sharing.

First, in an asynchronous network, online error correction and the fact that the clique is of size $n - t_a > 2t_s + \max(t_a, t_s - t_a) > 3t_a$ allows a party to reconstruct its correct polynomial by correcting at most t_a errors. On the other hand, we observe that if the network is synchronous, then all the honest parties' pairwise points get delivered to a party outside within a designated time which is known beforehand; however, we cannot ensure the correction of t_s errors. Here, we use the properties of the Reed-Solomon decoding algorithm, which allows a party to detect and correct errors simultaneously. A clever application of this technique, as discussed in the next paragraph, allows a party outside the clique to identify if the set of points it has received has more than t_a errors. This in turn allows the party to conclude if the network is synchronous leveraging the fact that $t_a < t_s$ holds. The knowledge that the network is synchronous allows a party outside the clique to conclude if the dealer behaves honestly or not, based on which it can either output the received univariate polynomial or \perp . We ensure that for a misbehaved corrupt dealer, it always outputs \perp .

We now conclude with the description of how the simultaneous error correction and detection is leveraged in our protocol. As mentioned, in a synchronous network it is guaranteed that a party receives at least $n - t_a - t_s \geq t_s + t_a + 1$ points from the honest parties in the clique. Moreover, these will be received within a designated time which is known beforehand for a

Number of points received	Correct	Detect	Outcome	
			Sync	Async
$t_s + t_a + 1$	0	t_a	Success	Wait
$t_s + t_a + 2$	1	$t_a - 1$	Success	Wait
\vdots	\vdots	\vdots	\vdots	\vdots
$t_s + 2t_a$	$t_a - 1$	1	Success	Wait
$t_s + 2t_a + 1$	t_a	0	Success	Success
$t_s + 2t_a + 2$	t_a	1	Detect	-
$t_s + 2t_a + 3$	t_a	2	Detect	-
\vdots	\vdots	\vdots	\vdots	\vdots
$2t_s + t_a + 1$	t_a	$t_s - t_a$	Detect	-

Table 6.1: Simultaneous error correction and detection

synchronous network. Hence, upon receiving $t_s + t_a + 1$ points, a party starts the decoding procedure. It then decides on the number of errors to be detected and corrected as per Table 6.1 and decides on whether to accept the reconstructed polynomial as indicated. Suppose a party outside the clique receives $m = t_s + t_a + 1 + x$ points from the parties in the clique. Let us analyze the scenario of a synchronous network. If $x \leq t_a$, then the decoding procedure is guaranteed to succeed due to the following: (i) at most x of the total m points are erroneous, and (ii) the number of errors that can be corrected equals $\frac{m - (t_s + 1)}{2} \geq x$. Hence, if the reconstruction succeeds, the party can output the reconstructed polynomial. On the other hand, if $x > t_a$, then by properties of the decoding algorithm, it can detect the presence of more than t_a errors and conclude that the network is asynchronous. Now consider the case of an asynchronous network when the party outside receives the same number of points $m = t_s + t_a + 1 + x$. Unlike the case of a synchronous network, we do not have the guarantee that at most x points are erroneous. Since the network is asynchronous and the messages are received in arbitrary or even adversarially controlled order, it is possible that there are up to t_a erroneous points. Hence, we need the mechanism of allowing for correction of up to x and additionally detection of up to $x - t_a$ errors simultaneously. In this case, if there indeed are more than x errors, then the reconstruction fails and the party can afford to receive more correct points from the slow honest parties. Note that in the worst case, when $x = t_a$, the reconstruction will definitely succeed. In our protocol, we use these observations to allow a party outside the clique to recover its polynomial. We refer the readers to Section 6.3 for more details about simultaneous error correction and detection and the exact bounds.

From weak secret sharing to verifiable secret sharing. We use the standard approach taken in the prior works [98, 97, 91, 13] to extend the weak secret sharing scheme to the stronger primitive of verifiable secret sharing. For this, we rely on a “two-layer” approach, wherein the first layer is similar to the weak secret sharing, whereas the second layer enables parties outside the clique to recover their polynomial even when the dealer is corrupt in the synchronous network. More specifically, in the verifiable secret sharing protocol, parties proceed very similarly to weak secret sharing, however, the pairwise consistency checks are now performed differently. Instead of directly exchanging their pairwise points, each party now initiates an instance of weak secret sharing to share its univariate polynomial received from the dealer. A party broadcasts $OK(j)$ for a party P_j in the verifiable secret sharing if and only if it computes the pairwise point as output in P_j 's instance of weak secret sharing. Doing so allows a party outside the clique to reconstruct its correct polynomial based on the points from parties in whose weak secret sharing instances it computes an output. This is a standard technique to extend weak secret sharing to verifiable secret sharing, and we refer the readers to [13] and the proof of our protocol for more details of its correctness.

Challenges in achieving polynomial time protocol. We now briefly discuss the challenges we encountered while trying to achieve a polynomial time algorithm for weak secret sharing. Note that the only exponential time component in our protocol is that of clique finding of size $n - t_s$. Specifically, we allow the dealer to run in exponential time and identify a clique of size $n - t_s$. We stress that identifying such a clique is crucial to allow for its extension to size $n - t_a$. Recall that during the clique extension phase if the dealer cannot restart the protocol by adding a new party to U , then it approaches the clique extension via an alternative technique. Here, the dealer identifies a set V of (at most) $t_s - t_a$ parties outside the clique to instruct parties to resolve all their inconsistencies. Suppose for the purpose of this discussion that the dealer has identified a clique of size $n - t_s$ and the set V is of size $t_s - t_a$. If all the parties successfully broadcast consistency with parties in V and vice versa, then we are guaranteed that the parties in V are now consistent with all the parties in the clique. Hence, we now have that guarantee that the consistency graph indeed has a clique of size $(n - t_s) + (t_s - t_a) = n - t_a$. However, the same does not hold true if the parties in V are not carefully chosen from those outside of an $n - t_s$ sized clique. If the set V is chosen to be an arbitrary set of parties, then ensuring that they are consistent with all the parties does not suffice to guarantee a bigger sized clique in the consistency graph. This is because the $t_s - t_a$ parties from V may already be included in the clique. Although they are now consistent with all parties outside the clique, the other parties from the clique may still have inconsistencies with those outside, preventing clique expansion. This is the precise reason why we required identifying the exact clique in

our protocol. We leave it as an interesting direction to identify if clique expansion can occur without requiring clique finding, for instance by using techniques such as Star algorithm [40].

6.2.2 Verifiable Triple Sharing

In a verifiable triple sharing (VTS) protocol, the dealer is required to share a multiplication triple *verifiably* while ensuring privacy of the triple. Our starting point is the verifiable triple sharing schemes of [50] which are designed independently for both the synchronous and the asynchronous networks. We outline their synchronous protocol with $t_s < n/3$ assuming a synchronous verifiable secret sharing scheme, which outputs t_s -sharing of the input secret. This is followed by the slight changes needed for their asynchronous verifiable triple sharing.

To share a multiplication triple, the dealer first chooses $2t_s + 1$ random multiplication triples (a_i, b_i, c_i) for $i \in \{1, \dots, 2t_s + 1\}$ and shares them via degree- t_s polynomials using the verifiable secret sharing protocol. To verify the multiplicative relation, parties first transform these random triples into correlated triples (x_i, y_i, z_i) such that they lie on polynomials X, Y, Z of degree $t_s, t_s, 2t_s$ respectively such that $XY = Z$ if and only if all the input triples (a_i, b_i, c_i) for $i \in \{1, \dots, 2t_s + 1\}$ are multiplication triples. Therefore the task of verifying the input triples reduces to the task of verifying $XY = Z$. Towards the latter, the sharings of $X(i), Y(i), Z(i)$, i th point on each of these polynomials is reconstructed to *only* P_i , who locally verifies that $X(i) \cdot Y(i) = Z(i)$ holds and broadcasts the result of its verification. If the verification fails for some party P_i , then parties publicly reconstruct $X(i), Y(i), Z(i)$ and verify the relation. If it fails, then the dealer is discarded. Otherwise, the protocol completes successfully if the (local or public) verification holds for at least $3t_s + 1$ parties, which in turn includes at least $2t_s + 1$ honest parties. The latter confirms that $XY = Z$, since the polynomials are of degree at most $2t_s$. The output of parties is the sharing of $X(\beta), Y(\beta), Z(\beta)$ for some public value $\beta \notin \{1, \dots, n\}$. In the above protocol, the degree of the polynomials X and Y is crucially defined to t_s to ensure privacy and correctness of triple verification. Observe that the verification process reveals one point on these polynomials to every party, allowing the adversary to learn (at most) t_s points. Setting a smaller degree would allow an adversary to obtain the complete polynomials X, Y , violating the privacy of the output triple $X(\beta), Y(\beta), Z(\beta)$ for an honest dealer. On the other hand, having a higher degree would not ensure verification of a corrupt dealer's triples. Consider the scenario when $n = 3t_s + 1$. If the polynomials X, Y are of degree d such that $d > t_s$, then consequently, Z will be of degree more than $2d > 2t_s + 1$. This requires at least $2d + 1 > 2t_s + 2$ points on the polynomial to be verified by the *honest* parties, which is not possible since there may be only $2t_s + 1$ honest parties in the network in the worst case. We summarize the synchronous verifiable triple

sharing scheme of [50] below.

1. The dealer shares $2t_s + 1$ multiplication triples, say (a_i, b_i, c_i) for $i \in \{1, \dots, 2t_s + 1\}$ through a verifiable secret sharing protocol.
2. Parties transform these triples into correlated triples such that they lie on polynomials $X(\cdot), Y(\cdot), Z(\cdot)$ where $X(\cdot) \cdot Y(\cdot) = Z(\cdot)$ holds. Specifically, parties define the polynomials $X(\cdot), Y(\cdot), Z(\cdot)$ of degree $t_s, t_s, 2t_s$ respectively such that $X(i) = a_i, Y(i) = b_i$ and $Z(i) = c_i$ for each $i \in \{1, \dots, t_s + 1\}$. Note that the degree t_s polynomials $X(\cdot)$ and $Y(\cdot)$ are completely defined by these points. Parties hold shares of each of these $t_s + 1$ points on the three polynomials.
3. Using linearity of t_s -sharing, parties hold sharing of the extrapolated points $X(i), Y(i)$ for every $i \in \{t_s + 2, \dots, 2t_s + 1\}$.
4. Given these, they now require to compute $Z(i)$ for every $i \in \{t_s + 2, \dots, 2t_s + 1\}$ while maintaining the multiplicative relation. For this, parties consume one multiplication triple (a_i, b_i, c_i) shared by the dealer and use Beaver's multiplication protocol to obtain the sharing of $Z(i) = X(i) \cdot Y(i)$ from the sharings of $X(i), Y(i)$ for each $i \in \{t_s + 2, \dots, 2t_s + 1\}$. Note that the polynomial $Z(\cdot)$ of degree $2t_s$ is now defined completely.
5. Using linearity on the sharings of $\{X(i), Y(i), Z(i)\}$ for $i \in \{1, \dots, 2t_s + 1\}$, parties obtain sharings of $X(i), Y(i), Z(i)$ for each $i \in \{2t_s + 2, \dots, n\}$ through local computation. Thus parties now have sharings of each $X(i), Y(i), Z(i)$ for $i \in \{1, \dots, n\}$.
6. To verify the multiplicative relation of the shared triples, parties have to ensure that $X(\cdot) \cdot Y(\cdot) = Z(\cdot)$ holds. Towards this, $X(i), Y(i), Z(i)$ are reconstructed to P_i , who verifies that $X(i) \cdot Y(i) = Z(i)$ holds and broadcasts the result of the verification, either OK or NOK, to all the parties. Note that this step leaks t_s points on polynomials X, Y, Z to the adversary when the dealer is honest.
7. For each party P_i whose verification fails, the check is performed publicly by reconstructing the points $X(i), Y(i), Z(i)$ to all.
8. Since the polynomials are of degree $t_s, t_s, 2t_s$ respectively, the triples are successfully verified if $2t_s + 1$ honest parties (and hence $3t_s + 1$ parties in total) confirm the relation. Otherwise, the dealer is discarded.

In the asynchronous setting with $t_a < n/4$, the protocol operates with the appropriate threshold t_a both for sharing as well as the degree of X, Y ; the rest of the steps follow closely to the synchronous case with a few caveats. For instance, to avoid an endless wait in the asynchronous setting, parties can afford to wait for the OK or NOK broadcast of at most

$n - t_a$ parties. However, given that $n - t_a \geq 3t_a + 1$ and the polynomials X, Y are now of degree t_a , correctness is ensured when the multiplicative relation is verified for $n - t_a$ parties. For an honest dealer, all the $n - t_a$ honest parties will eventually broadcast OK, ensuring that the triple sharing is successful. On the other hand, verifying $n - t_a$ points on the polynomial ensures correctness even for a corrupt dealer.

Network-agnostic protocol in the non-optimal threshold setting [13]. Recall that they use $n > 3t_s + t_a$. Being agnostic of the network style and the threshold, [13] follows the above protocol idea while keeping the degree of the sharings and X, Y as t_s (recall that $t_s > t_a$) and makes sure that $X(i) \cdot Y(i) = Z(i)$ holds for at least $2t_s + 1$ honest parties as follows. They define a set W of parties with $|W| \geq n - t_s$ and make sure that every party in W verifies $X(i) \cdot Y(i) = Z(i)$ either privately or publicly. The set W is constructed such that it contains all the $n - t_s$ honest parties when the network is synchronous and it contains at least $2t_s + 1$ honest parties when the network is asynchronous. For this, they wait till a designated time and add to W the first (at least) $n - t_s$ parties which respond to the verification of dealer's triples. The designated time is such that in the synchronous network, all the honest parties respond within this time and hence get included in W . On the other hand, in the asynchronous network W may consist of arbitrary $n - t_s$ parties depending on the scheduling of messages of these parties. Hence, W may contain t_a corrupt parties, resulting in the presence of at least $|W| - t_a$ honest parties inside W .

As mentioned, they ensure that every party in W verifies $X(i) \cdot Y(i) = Z(i)$ either privately or publicly. This works when the network is synchronous, since every honest party is in W and there are at least $2t_s + 1$ honest parties. A corrupt dealer will get caught if it shares triples that are not multiplication triples. In contrast, when the network is asynchronous, they have at least $|W| - t_a \geq 2t_s + 1$ honest parties in W , which again ensures that either the triples are correct or the dealer is discarded.

Challenges with optimal-threshold. We observe that the protocol of [13] crucially relies on the resilience of $n > 3t_s + t_a$ to ensure correctness of triples. Specifically, reducing the threshold to optimal has an immediate problem in ensuring that the triples shared indeed satisfy the multiplicative relation. When $n > 2t_s + \max(2t_a, t_s)$, we have that $|W| = n - t_s \geq t_s + \max(2t_a, t_s) + 1$. Assume that the network is asynchronous. It no longer holds that $|W| - t_a \geq 2t_s + 1$. Hence, the correctness of the triples cannot be established. Further, expecting a bigger W , say of size $n - t_a$ to ensure the correctness may result in an indefinite wait even for an honest dealer. This is because, an adversary corrupting up to t_s parties may remain silent, preventing the protocol from proceeding.

Extending to the network-agnostic setting with optimal resilience. We now discuss our techniques, which extend the ideas of the above approach to the network-agnostic setting. To account for the worst-case corruption threshold, our protocol too operates with t_s -sharing and polynomials X, Y of degree t_s . Observe that, following a similar template as above, to ensure the correctness of the multiplicative relation, $2t_s + 1$ honest parties are required to confirm their local verification, had the network been synchronous. On the other hand, in the asynchronous setting, it suffices for *any* $t_a + (2t_s + 1)$ parties to confirm.

We ensure these two conditions hold in our network agnostic protocol as follows. First we enforce that parties resolve the NOK received from any party within a pre-specified time before computing their output in the protocol. Second, we demand that the total number of distinct points i for which $X(i) \cdot Y(i) = Z(i)$ is verified, either privately or publicly, be at least $n - t_a$. Contrast this with the $n - t_s$ number of points required to be verified in [13]. The first requirement ensures correctness in the synchronous network, whereas the second condition guarantees it in the asynchronous network. Specifically, in a synchronous network, the properties offered by the network-agnostic broadcast protocol make sure that all the honest parties receive the OK or NOK messages from other honest parties within a designated time. Hence, they compute their output only upon verifying each NOK message received. This ensures that if the dealer is not discarded, then $X(i) \cdot Y(i) = Z(i)$ has been verified for all the honest parties. Since there are at least $2t_s + 1$ honest parties in the synchronous case, we are guaranteed correctness of the triples. On the other hand, if the network is asynchronous, the second condition of verifying a total of $n - t_a$ points comes into effect to ensure correctness. Since parties verify the multiplicative relation for $n - t_a$ points and the adversary can corrupt at most t_a parties, we have that the relation holds for at least $n - 2t_a \geq 2t_s + 1$ honest parties. Again, we are guaranteed correctness of the multiplication triples. However, making sure these two conditions hold requires additional techniques.

Observe that in a synchronous network, we can expect at most $n - t_s$ parties to broadcast the result of their local verification within the designated time. There may be t_s corrupt parties which remain silent, that is, these parties neither broadcast OK nor NOK. In such a case, enforcing a support of $n - t_a$ would result in stalling the protocol even for an honest dealer. To remedy this, we perform a *dealer-guided* public reconstruction of points $X(i), Y(i), Z(i)$ of a subset of parties who either broadcast NOK later than the designated time or are silent. We enforce that the total number of points verified, which includes the publicly verified values and those from the OK messages of parties is at least $n - t_a$. Here, the term *dealer-guided* refers to the criteria that the dealer chooses the parties whose points have to be reconstructed publicly. An important aspect to note here is that with t_s -degree polynomials we do not have any

additional degree of freedom in the synchronous setting. Moreover, we have only $t_s - t_a$ degree of freedom in the asynchronous setting. Thus, revealing the points has to be performed carefully by the dealer. To maintain privacy here, we ensure that the dealer only begins the guided reconstruction upon waiting for a designated time and additionally receiving *at least* $n - t_s$ OKs, and performs the public reconstruction for *at most* $t_s - t_a$ parties. The intuition behind privacy in the synchronous setting is that honest parties always broadcast their OK messages which are received by all within in the designated time. Hence, their points are never reconstructed publicly. On the other hand, in the asynchronous setting, some of the honest parties may be slow. However, an honest dealer reveals points for at most $t_s - t_a$ honest parties, still ensuring the degree of freedom of 1 and hence maintaining privacy.

In conclusion, our protocol ensures that parties verify all the NOK received within a designated time and that the verification succeeds for at least $n - t_a \geq t_a + 2t_s + 1$ parties in total. This ensures that if the dealer is not discarded, the triples generated are correct regardless of the underlying network type.

6.2.3 Putting it all together: The MPC Protocol

We now describe the construction of our MPC protocol using the above primitives. At a very high level, the MPC protocol uses Beaver’s circuit randomization trick [23] and adopts a two-phase structure. The first phase corresponds to *Beaver triple generation*, followed by the second phase of *circuit evaluation*. The verifiable secret sharing and verifiable triple sharing primitives are utilized in the former phase, whereas existing primitives suffice for the latter.

Beaver triple generation. In more detail, the Beaver triple generation phase ensures that verified random multiplication triples are shared among parties as follows. Each party enacts the role of the dealer and shares random multiplication triples using verifiable secret sharing. Upon completing the sharing, parties must verify the correctness of these triples, which is precisely where they rely on the verifiable triple sharing. If the triples shared by a dealer are verified to be correct, then they are accepted in the further computation. Otherwise, parties discard the dealer and assume a default sharing on behalf of the dealer. Note that the corrupt parties may never initiate an instance of triple sharing. Moreover, given that the network may be asynchronous, waiting for all n dealers’ instances may result in an endless wait. To prevent this, we use an instance of a primitive called asynchronous common set (ACS), which allows parties to agree on a common set of at least $n - t_s$ dealers whose shared triples will be used in circuit evaluation. Agreement on this set amongst parties is also crucial since it is likely that different parties compute their output in the triple sharing instances of dealers in a different order due to asynchrony. At this stage, we have that parties have agreed on the set of triples

shared by at least $n - t_s$ dealers. To ensure secure evaluation of the circuit, we however require random multiplication triples that are unknown to any party. For this, we use a ‘triple extraction’ protocol from the literature [50, 13] which consumes one triple shared by each dealer and extracts a random triple unknown to any party. This completes the Beaver triple generation phase, with parties holding shares corresponding to random multiplication triples.

Circuit evaluation. Parties use the triples from the prior phase to perform a shared evaluation of the circuit. Parties begin by sharing their inputs to the circuit. Similar to the case of triple sharing, waiting for the input of all the parties may result in an endless wait. Hence, parties run an instance of ACS to agree on a set of at least $n - t_s$ parties whose input will be considered for evaluation. A default value is assumed as the input of the remaining parties. We remark that in our protocol, parties can simultaneously perform triple sharing and input sharing, followed by a single instance of the ACS protocol. Subsequently, the evaluation of the circuit proceeds as follows. The evaluation of linear gates (addition and multiplication by a constant) is done locally. Parties use one multiplication triple from the first phase and rely on Beaver’s multiplication [23] protocol to evaluate a multiplication gate. Following this, parties reconstruct the protocol output. Finally, they terminate upon ensuring that a sufficient number of parties have computed the output so as to ensure that all parties obtain their output. This concludes our MPC protocol.

6.3 Preliminaries

6.3.1 Network Model and Definitions

We consider a set of parties $\mathcal{P} = \{P_1, \dots, P_n\}$ connected via pairwise private and authenticated channels. The distrust among the parties is modeled as a centralized, *computationally unbounded* adversary. We consider a static adversary that decides the set of corrupt parties at the beginning of the protocol execution. The underlying network conditions can be synchronous or asynchronous, and the parties are unaware of the exact network type during the protocol execution. In a synchronous network, every message sent is delivered within a fixed, known time bound Δ . Moreover, the messages are delivered in the same order they are sent in. In contrast, in the asynchronous network, the messages are delivered with an arbitrary but finite delay with the only guarantee that the messages are eventually delivered. Moreover, the messages may be delivered in an arbitrary order. This is modeled by a scheduler which decides on the sequence of message deliveries, where the scheduler is assumed to be controlled by the adversary. The adversary can corrupt up to t_s out of the n parties maliciously when the network is synchronous, whereas it can corrupt up to t_a parties under

asynchronous network conditions and make them behave arbitrarily.

Our protocols are defined over a field \mathbb{F} , such that $|\mathbb{F}| > n$. We denote the elements of the field by $\{0, 1, \dots, n\}$. Further, we use $\langle v \rangle$ to denote the degree- t_s Shamir-sharing of a value v among parties in \mathcal{P} .

Additionally, in constructing our protocols, we use several well-known primitives from the literature. We elaborate on these in Section 6.4 and refer the readers to the same for further details.

6.3.2 Symmetric Bivariate Polynomials

A degree (l, l) symmetric bivariate polynomial over \mathbb{F} is of the form $F(x, y) = \sum_{i,j=0}^{l,l} b_{ij}x^i y^j$ where $b_{ij} \in \mathbb{F}$ and $b_{ij} = b_{ji}$ holds for all $i, j \in \{0, \dots, l\}$. This implies that $F(i, j) = F(j, i)$ holds for every i, j . Moreover, $F(x, i) = F(i, y)$ is also true for each $i \in \{1, \dots, n\}$.

Our protocol uses (t_s, t_s) symmetric bivariate polynomials. Further, $f_i(x) = F(x, i) = F(i, y)$ is called the i th univariate polynomial of $F(x, y)$ and is associated with party P_i in the protocol.

6.3.3 Finding a (n, t) -Star

Definition 6.3.1. Let G be a graph over the nodes $\{1, \dots, n\}$. We say that a pair (C, D) of sets such that $C \subseteq D \subseteq \{1, \dots, n\}$ is an (n, t) -star in G if the following hold: (a) $|C| \geq n - 2t$, (b) $|D| \geq n - t$, (c) For every $j \in C$ and every $k \in D$, the edge (j, k) exists in G .

6.3.4 Almost-surely Terminating

Following the approach of Appan et al. [13], we use randomized asynchronous byzantine agreement protocols designed for threshold $t_a < t_s < n/3$ (note that our resiliency matches with this requirement) in our work, which guarantee that *almost-surely* all the honest parties eventually receive their output. This implies that the probability that an honest party receives its output after participating in an infinite number of rounds of a protocol approaches 1 asymptotically [3, 88, 22]. Specifically,

$$\lim_{T \rightarrow \infty} \Pr [\text{An honest party } P_i \text{ receives its output by local time } T] = 1$$

where the probability is over the randomness of the honest parties and the adversary in the protocol. Also, the property of almost-surely receiving the output carries forward to all the protocols that use asynchronous byzantine agreement as a primitive. Similar to [13], for simplicity, we do not specify the terminating condition for each sub-protocol. Rather, when a party terminates the MPC protocol, it also terminates in all the sub-protocol instances.

6.3.5 Simultaneous Error Correction and Detection of Reed-Solomon Codes

We require the following coding-theory related results. Let C be a Reed-Solomon (RS) code word of length N , corresponding to a k -degree polynomial (containing $k + 1$ coefficients). Assume that at most t errors can occur in C . Let \bar{C} be the word after introducing error in C in at most t positions. Let the distance between C and \bar{C} be s where $s \leq t$. Then there exists an *efficient* decoding algorithm that takes \bar{C} and a pair of parameters (e, e') as input, such that $e + e' \leq t$ and $N - k - 1 \geq 2e + e'$ hold and gives one of the following as output:

1. Correction: output C if $s \leq e$, i.e. the distance between C and \bar{C} is at most e ;
2. Detection: output “more than e errors” otherwise.

Note that detection does not return the error indices; rather, it simply indicates error correction fails due to the presence of more than correctable (i.e., e) errors. The above property of RS codes is traditionally referred to as *simultaneous error correction and detection*. In fact, the bounds, $e + e' \leq t$ and $N - k - 1 \geq 2e + e'$, are known to be necessary. We cite:

Theorem 6.3.2 ([49, 87]). *Let C be a Reed-Solomon (RS) code word of length N , corresponding to a k -degree polynomial (containing $k + 1$ coefficients). Let \bar{C} be a word of length N such that the distance between C and \bar{C} is at most t . Then RS decoding can correct up to e errors in \bar{C} to reconstruct C and detect the presence of up to $e + e'$ errors in \bar{C} if and only if $N - k - 1 \geq 2e + e'$ and $e + e' \leq t$.*

Corollary 6.3.3. *Let C and \bar{C} be as in Theorem 6.3.2 with $N = t_s + t_a + 1 + x$, $k = t = t_s$ and $x \leq t_a$. Then RS decoding can correct up to x errors and detect the presence of up to $t_a - x$ errors in \bar{C} .*

Proof. This follows since $N - k - 1 = t_a + x$, $2e + e' = 2x + (t_a - x) = t_a + x$ and $e + e' = t_a < t_s$ hold. □

Corollary 6.3.4. *Let C and \bar{C} be as in Theorem 6.3.2 with $N = t_s + t_a + 1 + x$, $k = t = t_s$ and $t_a < x \leq t_s$. Then RS decoding can correct up to t_a errors and detect the presence of up to $x - t_a$ errors in \bar{C} .*

Proof. This follows since $N - k - 1 = t_a + x$, $2e + e' = 2t_a + (x - t_a) = t_a + x$ and $e + e' = x \leq t_s$ hold. □

6.4 Existing Primitives

In our work, we use network-agnostic protocols from [13] for several primitives, such as broadcast and byzantine agreement, to name a few. Although designed for the non-optimal threshold, these naturally follow to the optimal threshold scenario. Below, we give a description of each of them along with the (n, t) -Star algorithm from [40] for completeness.

6.4.1 Finding a (n, t_a) -Star

Definition 6.4.1. *Let G be a graph over the nodes $\{1, \dots, n\}$. We say that a pair (C, D) of sets such that $C \subseteq D \subseteq \{1, \dots, n\}$ is an (n, t) -star in G if the following hold:*

- $|C| \geq n - 2t$,
- $|D| \geq n - t$,
- For every $j \in C$ and every $k \in D$, the edge (j, k) exists in G .

Canetti [39] showed that if a graph has a clique of size $n - t$, then there exists an efficient algorithm which always finds an (n, t) -Star. In our protocol, we use the parameter $t = t_a$. For completeness, we describe the algorithm for finding an (n, t_a) -Star in Algorithm 6.4.2, which is taken from [30, 40] and modified to suit our parameter. Moreover, we modify the algorithm from [40] to output the extended Star using the techniques of [92].

Protocol 6.4.2: (n, t_a) -Star

Input: An undirected graph G (over the nodes $\{1, \dots, n\}$) and a parameter t_a .

1. Find a maximum matching M in \overline{G} . Let N be the set of matched nodes (namely, the endpoints of the edges in M) and let $\overline{N} := \{1, \dots, n\} \setminus N$.
2. Let T be the set of triangle-heads, i.e., all vertices that are not endpoints of the matching but they have two neighbors in the matching.

$$T := \{i \in \overline{N} \mid \exists j, k \text{ s.t. } (j, k) \in M \text{ and } (i, j), (i, k) \in \overline{G}\} .$$

Let $C := \overline{N} \setminus T$.

3. Let B be the set of matched nodes that have neighbors in C . That is, set:

$$B := \{j \in N \mid \exists i \in C \text{ s.t. } (i, j) \in \overline{G}\} .$$

Let $D := \{1, \dots, n\} \setminus B$.

4. If $|C| \geq n - 2t_a$ (i.e. $|C| \geq 2t_s + 1$) and $D \geq n - t_a$ (i.e. $|D| \geq 2t_s + t_a + 1$) then compute E as the set of all the parties which do not have edges with at least $2t_s + 1$ parties in C . Finally, construct a set F as the set of all the parties that do not have edges with at least $2t_s + 1$ parties in E .
 5. **Output:** If $|E| \geq n - t_a$ and $|F| \geq n - t_a$ then output (C, D, E, F) . Otherwise, output \perp .
-

6.4.2 Asynchronous Reliable Broadcast (Acast)

As in [13], we use Bracha's asynchronous reliable broadcast protocol [37] (also referred to as Acast) where there is a designated sender who holds a message $m \in \{0, 1\}^\ell$ to be communicated to all the parties. Appan et al. [13] demonstrated that the Acast protocol, although designed for the asynchronous network, also provides certain guarantees in the synchronous network. We recall the protocol and its properties below.

Protocol 6.4.3: Π_{Acast}

Input: The sender holds a message $m \in \{0, 1\}^\ell$.

1. The sender on holding an input m , sends (init, m) to all the parties.
 2. Upon receiving (init, m) from the sender, send (echo, m) to all the parties. Do not execute this step more than once.
 3. Upon receiving (echo, m') from $n - t$ parties, send (ready, m') to all the parties.
 4. Upon receiving (ready, m') from $t + 1$ parties, send (ready, m') to all the parties.
 5. Upon receiving (ready, m') from $n - t$ parties, output m' .
-

Lemma 6.4.4. *Bracha's Acast protocol Π_{Acast} is secure against an adversary corrupting up to $t < n/3$ parties and achieves the following properties.*

1. *Synchronous Network:*
 - (a) *Liveness:* If the sender is honest, then all the honest parties obtain an output within time 3Δ .
 - (b) *Validity:* If the sender is honest, then every honest party with an output, has the sender's message m as the output.
 - (c) *Consistency:* If the sender is corrupt and some honest party outputs m' at time T , then every honest party outputs m' within time $T + 2\Delta$.

2. *Asynchronous Network:*

- (a) *Liveness:* If the sender is honest, then all honest parties eventually obtain an output.
- (b) *Validity:* If the sender is honest, then every honest party with an output, has the sender's message m as the output.
- (c) *Consistency:* If the sender is corrupt and some honest party outputs m' , then every honest party eventually outputs m' .

6.4.3 Byzantine Broadcast (BC)

Appan et al. [13] construct a broadcast protocol which relies on Bracha's asynchronous reliable broadcast [37] and an existing synchronous byzantine agreement protocol which is denoted by Π_{SBA} . We give the protocol Π_{BC} for broadcast below, assuming the existence of Π_{Acast} and Π_{SBA} . We avoid repetition and refer the readers to [13] for further details on the exact instantiation of these protocols since we make a black-box use of these primitives.

Protocol 6.4.5: Π_{BC}

Input: The sender holds a message $m \in \{0, 1\}^\ell$.

(Regular Mode):

1. The sender Acasts the message m using Π_{Acast} .
2. **At time 3Δ ,** each party P_i participates in an instance of synchronous broadcast protocol Π_{SBA} with its input set as follows:
 - If $m' \in \{0, 1\}^\ell$ is received from the Acast of the sender, then P_i sets m' as the input.
 - Otherwise, P_i sets its input as \perp .
3. **At time $3\Delta + T_{\text{SBA}}$,** each P_i computes its output as follows:
 - If $m' \in \{0, 1\}^\ell$ is received from the Acast of the sender and m' is computed as the output of Π_{SBA} , then P_i sets m' as the output.
 - Otherwise, P_i sets its output as \perp .

(Fallback Mode):

1. Each P_i which has computed its output as \perp at time $3\Delta + T_{\text{SBA}}$, updates it to m' if m' is received from the Acast of the sender.
-

Lemma 6.4.6. *Protocol Π_{BC} is secure against an adversary corrupting up to $t < n/3$ parties and has the following properties, where $T_{\text{BC}} = 3\Delta + T_{\text{SBA}} = (12n - 3)\Delta$ when Π_{SBA} is instantiated using [33].*

1. Synchronous network:

(a) (Regular Mode)

- i. Liveness: At time T_{BC} , every honest party has an output (through regular-mode).
- ii. Validity: If the sender is honest, then every honest party outputs m (through regular-mode).
- iii. Consistency: If the sender is corrupt, then every honest party has the same output (m' or \perp) at the end of T_{BC} (through regular-mode).

(b) (Fallback Mode)

- i. Fallback Consistency: If the sender is corrupt and some honest party outputs m' at time $T > T_{BC}$ (through fallback-mode), then every honest party outputs m' by time $T + 2\Delta$ (through fallback-mode).

2. Asynchronous network:

(a) (Regular Mode)

- i. Liveness: At time T_{BC} , every honest party has an output (through regular-mode).
- ii. Weak Validity: If the sender is honest, then every honest party outputs m or \perp (through regular-mode).
- iii. Weak Consistency: If the sender is corrupt, then every honest party has either a common m' or \perp as the output at the end of T_{BC} (through regular-mode).

(b) (Fallback Mode)

- i. Fallback Validity: If the sender is honest, then each honest party that outputs \perp at T_{BC} (through regular-mode) outputs m (through fallback-mode).
- ii. Fallback Consistency: If the sender is corrupt and some honest party outputs m' at time T (either through regular or fallback-mode), then every honest party eventually outputs m' (either through regular or fallback-mode).

6.4.4 Byzantine Agreement (BA)

Appan et al. [13] provide a network-agnostic byzantine agreement protocol by following the approach of [34]. Here, each party first broadcasts its input via an instance of Π_{BC} followed by running an instance of some asynchronous byzantine agreement protocol Π_{ABA} . Each party decides its input to Π_{ABA} based on the number of parties for which it received an output in their respective instance of the broadcast protocol and the plurality of the received values. We provide the protocol from [13] below for completeness, where Π_{ABA} can be instantiated with any existing protocol such as [3, 22].

Protocol 6.4.7: Π_{BA}

Input: Each P_i holds a bit $b_i \in \{0, 1\}$. Each P_i also initialises a set $R_i \leftarrow \phi$.

1. Each P_i on holding an input b_i , broadcasts b_i using Π_{BC} .
 2. For $j \in \{1, \dots, n\}$, let $b_i^{(j)} \in \{0, 1, \perp\}$ be received from the broadcast of P_j via regular mode. Update $R_i = R_i \cup \{j\}$ if $b_i^{(j)} \neq \perp$. Compute the input v_i for an instance of Π_{ABA} as follows:
 - If $|R_i| \geq n - t$ then set v_i to be the majority bit among the $b_i^{(j)}$ values of parties in R_i . If there is no majority, then set $v_i = 1$.
 - Otherwise, set $v_i = b_i$.
 3. At time T_{BC} , participate in an instance of Π_{ABA} with input v_i . Set the output as the output computed from Π_{ABA} .
-

Lemma 6.4.8. *Protocol Π_{BA} achieves the following properties in the presence of an adversary which corrupts up to $t < n/3$ parties:*

1. *Synchronous network: The protocol is a perfectly-secure SBA protocol, where all the honest parties receive their output within time $T_{BA} = T_{BC} + T_{ABA}$.*
 - (a) *Guaranteed liveness: All the honest parties obtain an output by time T_{BA} .*
 - (b) *Validity: If all the honest parties have the same input v , then all the honest parties with an output, outputs v .*
 - (c) *Consistency: All the honest parties with an output, output the same value v .*
2. *Asynchronous network: The protocol is a perfectly-secure ABA protocol.*
 - (a) *Almost-surely liveness: Almost-surely, all the honest parties obtain an output eventually.*
 - (b) *Validity: If all the honest parties have the same input v , then all the honest parties with an output, outputs v .*
 - (c) *Consistency: All the honest parties with an output, output the same value v .*

6.4.5 Agreement on a Common Set (ACS)

The ACS primitive [40] allows parties to agree on a common set of at least $n - t$ parties $\text{Com} \subset \mathcal{P}$, such that each party in Com satisfies some predefined property prop which has the following features in the asynchronous network:

1. Every honest party eventually satisfies prop .

2. If some honest P_i sees that a party P_j satisfies prop, then eventually all the honest parties see that P_j satisfies prop.

Although the above protocol was primarily designed for the asynchronous network, it was shown in [13] that the protocol satisfies certain properties in the synchronous network where each party in Com satisfies some predefined property prop which has the following features:

1. Every honest party satisfies prop at the onset of the protocol.
2. If some honest P_i sees that a party P_j satisfies prop, then within a fixed time, all the honest parties see that P_j satisfies prop.

In our protocols, we use the parameter $t = t_s$. We describe the variant of the protocol from [40], which was used in [13] for completeness.

Protocol 6.4.9: Π_{ACS}

Input: Each party P_i holds a dynamically growing set S_i .

Input Guarantees:

- If the network is synchronous, then for an honest P_i , at the onset $j \in S_i$ for each honest P_j . Moreover, if a corrupt $k \in S_i$ for some honest P_i , then within a fixed time, $k \in S_j$ for all honest parties P_j .
 - If the network is asynchronous, then for an honest P_i , eventually $j \in S_i$ for each honest P_j . Moreover, if $k \in S_i$ for some honest P_i , then eventually $k \in S_j$ for all honest parties P_j .
1. Each P_i participates in an instance of byzantine agreement protocol Π_{BA}^j where $j \in \{1, \dots, n\}$ with input 1 if $j \in S_i$.
 2. Once (at least) $n - t_s$ instances of Π_{BA} terminate with output 1, P_i participates with input 0 in the byzantine agreement instances Π_{BA}^j such that $j \notin S_i$.
 3. Upon termination of all the n instances of byzantine agreement, P_i outputs Com as the set of parties P_j such that Π_{BA}^j terminated with the output 1.
-

Theorem 6.4.10. *Protocol Π_{ACS} is secure against an adversary corrupting up to t_s parties in the synchronous network and t_a parties in the asynchronous network and has the following properties.*

1. *Synchronous network:*

- (a) *Liveness*: At time $T_{\text{ACS}} = 2T_{\text{BA}}$, every honest party has an output.
- (b) t_s *correctness*: At time T_{ACS} , every honest party outputs Com of size at least $n - t_s$ such that the following holds:
- All the honest parties belong to Com .
 - For each $j \in \text{Com}$, it is guaranteed that $j \in S_i$ for each honest party P_i .

2. *Asynchronous network*:

- (a) *Liveness*: Almost-surely, every honest party eventually has an output.
- (b) t_a *correctness*: Almost-surely, every honest party eventually outputs Com of size at least $n - t_s$ such that the following holds:
- For each $j \in \text{Com}$, it is guaranteed that eventually $j \in S_i$ for each honest party P_i .

6.5 Lower Bound

Theorem 6.5.1. *For any n , if $2 \cdot \max(t_s, t_a) + \max(2t_a, t_s) \geq n$, then there is no n -party MPC protocol that is perfectly-secure against an adversary corrupting t_s parties in the synchronous network and t_a parties in the asynchronous network.*

Proof. We first consider two cases, when $t_s \leq t_a$ and otherwise. For the former case, we have that $4t_a \geq n$, and the known impossibility result of [30] follows immediately. For the latter scenario, when $t_s > t_a$, we further analyze it considering two cases. First, when $2t_a < t_s$, we have that $3t_s \geq n$. We now see that the impossibility of a network-agnostic protocol for this setting follows directly from the impossibility of synchronous protocols with this threshold [29]. Thus, what remains to be shown is the case of $2t_s + 2t_a \geq n$ when $t_s > t_a$ and $2t_a \geq t_s$. We prove this by contradiction as follows.

Assume $2t_s + 2t_a = n$, and there exists a generic MPC protocol π which is t_s -secure in the synchronous network and t_a -secure in the asynchronous network. Partition the n parties into four disjoint sets S_1, S_2, S_3, S_4 such that $|S_1| = |S_2| = t_s$ and $|S_3| = |S_4| = t_a$ and consider the following scenarios.

Case I: Synchronous network, Parties in S_1 (S_2) are corrupted. The adversary blocks all communication from parties in S_1 (S_2) towards parties in S_2 (S_1). Further, it ignores the messages received from the parties in S_2 (S_1) during its local computation. It performs the rest of the computation and communication as per the protocol specification.

Case II: Asynchronous network, Parties in S_4 are corrupted. In this case, the adversary indefinitely delays all the communication between the (honest) parties in S_1 and S_2 . The

adversary performs the computation and communication with the parties as per the protocol specification.

Observe that the corruption scenarios described above are valid in the synchronous and asynchronous networks, respectively. Moreover, each party's view is identical in both scenarios, thus guaranteeing that the parties remain unaware of the network type when either of the aforementioned corruption occurs during the protocol. The security guarantees of the protocol ensure that, in either case, parties receive the output of the protocol. We leverage these observations to arrive at a contradiction.

Specifically, we show that given such an n -party generic MPC protocol, we can construct an MPC protocol for 4 parties, say P_1, \dots, P_4 where P_i emulates the parties in S_i . This new protocol is secure with respect to an adversary that either corrupts one of P_1, P_2 when the network is synchronous or corrupts one among P_3, P_4 when the network is asynchronous. Now consider an instance of the protocol amongst the four parties to compute the following functionality:

$$f(x_1, x_2, \perp, \perp) \rightarrow (x_1 \wedge x_2, x_1 \wedge x_2, \perp, \perp)$$

We show that it is impossible for the output receiving parties, P_1 and P_2 to have a unanimous output, thus showing the impossibility of the underlying n party network-agnostic protocol. Consider the scenario when the network is asynchronous, and the adversary corrupts the party P_4 . Further, the adversary follows the same (valid) strategy of blocking communication between parties as described in **Case II**, which implies blocking communication between P_1 and P_2 in the 4-party protocol.

Let $\pi(x_1, x_2)$ be an instance of the protocol with inputs x_1, x_2 and $r_i^{\pi(x_1, x_2)}$ for each $i \in [4]$ denote the randomness of each P_i in the instance $\pi(x_1, x_2)$. Let T_{ij} ($1 \leq i < j \leq 4$) denote the transcript of the channels between P_i and P_j . Note that $T_{12} = \phi$. Moreover, due to perfect security, T_{13} and T_{14} individually are independent of P_1 's input x_1 . Otherwise, a corrupt P_3 or P_4 will be able to learn P_1 's input. For the same reason, T_{23} and T_{24} individually are independent of P_2 's input x_2 . Hence, we can conclude that P_1 's output is determined by its internal state and the joint distribution $\{T_{13}, T_{14}\}$. Similarly, P_2 's output is determined by its internal state and the joint distribution $\{T_{23}, T_{24}\}$. Suppose these are the transcripts of the protocol instance $\pi(0, 1)$. Since T_{23} and T_{24} are individually independent of x_1 , there exists some T'_{24} such that $\{T_{23}, T'_{24}\}$ results in an output 1 for P_2 . If not, then it implies that irrespective of T_{24} , the output of P_2 is always 0. This further implies that the output of P_2 is completely decided by its internal state and T_{23} . However, T_{23} itself is independent of x_1 . This is because, the view of P_3 in the protocol must be independent of x_1 , due to perfect security

and T_{23} is contained in the view of P_3 . Now note that P_1 does not communicate with P_2 at all. This means P_1 's input is ignored in the output computation of P_2 , leading to breach of correctness. Therefore, we can conclude that in an instance $\pi(0, 1)$, there exists some T'_{24} such that $\{T_{23}, T'_{24}\}$ results in an output 1 for P_2 .

Relying on the above fact, we can now conclude that an adversary corrupting P_4 in an instance of $\pi(0, 1)$ can behave according to T_{14}, T_{34} with P_1, P_3 respectively, and according to T'_{24} with P_2 . This results in P_1 having the output 0, while P_2 outputs 1. □

6.6 Weak Secret Sharing

Some part of the protocol proceeds in a sequence of time steps. Whereas some parts are action-based, parties execute these steps as and when they receive the messages required to perform these steps. Throughout the descriptions of protocols, we denote the wait period of time steps with **red** font, whereas the action-based steps of our protocols are denoted using **blue** font. We also use the existing primitives such as broadcast and agreement, which are emulated using Protocol 6.4.5 and Protocol 6.4.7 of [13] (recalled in Appendix 6.4.3 and 6.4.4 respectively). In the subsequent description, Δ denotes the round delay associated with a synchronous network. We also use the notations T_{BC} and T_{BA} to denote the time required by the broadcast and agreement protocols of [13] in the synchronous network. The exact values for these are inherited from their work and detailed in Section 6.4.

Protocol 6.6.1: Π_{WSS}

Input: The dealer holds a secret $s \in \mathbb{F}$.

Initialisation: The dealer initialises two sets W, U to \emptyset . Only W is reset in every (re)run to \emptyset . The set U is initialised *only* during the first run, and is used without re-initialisation during subsequent reruns.

1. (Polynomial Share Distribution) The dealer chooses a symmetric bivariate polynomial $F(x, y)$ of degree t_s in both x, y and delivers $f_i(x) = F(x, i)$ to P_i . If $|U| > t_s - t_a$, then assign U to be the set of first $t_s - t_a$ parties lexicographically. The dealer broadcasts $(U, \{f_i(x)\}_{i \in U})$.
2. (Pair-wise exchange) At time Δ , if $f_i(x)$ is received then every P_i sends $f_{ij} = f_i(j)$ to every P_j .
3. (Pair-wise Consistency Check) **At time T_{BC}** ¹ P_i prepares a vector R_i of length n as follows

¹Had the network been synchronous, then we know that $T_{BC} > \Delta$. Hence, $f_i(x)$ and the dealer's broadcast, both initiated simultaneously, will be received by P_i by T_{BC} .

and broadcasts it. It sets $R_i[j] = \text{NR}$ for all j if any of the following happens:

- (a) it receives no $f_i(x)$
- (b) the dealer's broadcast results in \perp
- (c) some $f_j(x)$ in the broadcast $(U, \{f_i(x)\}_{i \in U})$ is of degree more than t_s
- (d) there are indices j, k such that $f_j(k) \neq f_k(j)$ in the broadcast $(U, \{f_i(x)\}_{i \in U})$

Otherwise, it sets R_i as follows. (1) if $P_j \in U$, then $R_i[j] = f_i(j)$ (2) if $P_j \notin U$, then set

(a) $R_i[j] = \text{NR}$ if no f_{ji} is received from P_j , (b) $R_i[j] = f_i(j)$ if f_{ji} is received from P_j and $f_i(j) \neq f_{ji}$, (c) $R_i[j] = \text{OK}$ otherwise.

4. (Asynchronous Pair-wise Consistency Checking) The parties execute the following steps as and when they receive the required values. On receiving the broadcast $(U, \{f_i(x)\}_{i \in U})$ and polynomial $f_i(x)$ from the dealer, every $P_i \notin U$ sends $f_{ij} = f_i(j)$ to every P_j and broadcasts AOK_j if (a) f_{ji} from $P_j \notin U$ is received and $f_i(j) = f_{ji}$ (b) $f_j(i)$ for $P_j \in U$ satisfies $f_i(j) = f_j(i)$.
5. (Restart or Clique Finding) **At time $2T_{\text{BC}}$** , the dealer puts $P_i \notin U$ in W if either happens (a) P_i 's broadcast of R_i resulted in \perp or (b) P_i 's broadcasted R_i has more than t_s NRs or (c) $R_i[j] \neq F(i, j)$ when $R_i[j] \neq \text{OK}$ and $R_i[j] \neq \text{NR}$.

The dealer makes a graph G with n vertices corresponding to n parties. There is an edge when $R_i[j] = R_j[i] = \text{OK}$. There is no edge if $R_i[j] = \text{NR}$ or $R_j[i] = \text{NR}$. The dealer finds a clique Q of size $n - t_s + |U|$ in the graph including U . If $|Q| \geq n - t_a$, then the dealer sets $Q_a = Q$ and broadcasts (sync, G, Q_a) . Otherwise, if $|W| > 0$, then the dealer sets $U = U \cup W$ and broadcasts $(\text{restart}, U)$. Otherwise, it broadcasts $(\text{continue}, Q, G, V)$, where V is a set of $(t_s - t_a) - |U|$ parties (vertices) outside $Q \cup U$.

6. (Asynchronous Clique Finding) The dealer executes the following steps as and when it receives the required messages. First, the dealer initiates a graph A with parties as vertices with edges between a pair of parties in U . On receiving broadcasts AOK_{ij} and AOK_{ji} from $P_i, P_j \notin U$, it adds an edge between P_i, P_j . On receiving broadcast AOK_{ij} from $P_i \notin U, P_j \in U$, it adds an edge between P_i, P_j . Each time there is an update in A , it invokes $(C, D, E, F) \leftarrow \text{Star}(A)$ (Protocol 6.4.2) If $|F| > n - t_a$, it sets $Q_a = F$ and broadcasts (async, A, Q_a) .
7. (Conflict Resolution for Clique Expansion or Restart) **At time $3T_{\text{BC}}$** , the parties do the following:

- (a) If (sync, G, Q_a) is received, then P_i verifies G, Q_a as follows. It checks the validity of G_i in the same way as in Step 7c. It checks if Q_a is a $(n - t_a)$ -size clique in G_i including parties in U . If the verification passes, then set $b_i = 1$ and $b_i = 0$

otherwise and participate in an instance of Π_{BA} . If the protocol output is 1 then go to Protocol 6.6.2. Otherwise, wait for (async, A, Q_a) from the dealer.

- (b) If (restart, U) is received, then set $b_i = 0$ and participate in an instance of Π_{BA} . If the output is 1, then go to Protocol 6.6.2. Otherwise, restart the protocol from Step 1.
- (c) If (continue, Q, G, V) is received, then set $b_i = 0$ and participate in an instance of Π_{BA} . If the output is 1, then go to Protocol 6.6.2. Otherwise, when the output is 0, verify Q, G, V . For this, construct G_i exactly as the dealer did based on the broadcasts available at time $2T_{\text{BC}}$ at Step 5. G is marked as invalid if
 - i. it is different from G_i AND
 - ii. there is a pair $P_j, P_k \notin U$ such that $R_j[k] \neq R_k[j]$ or there is a pair $P_j \notin U, P_k \in U$ such that $R_j[k] \neq f_k(j)$.

Q is invalid if it is not a clique in a valid G of size at least $n - t_s + |U|$ and does not include parties in U . V is invalid if it is not a set of $(t_s - t_a) - |U|$ parties (vertices) outside $Q \cup U$ in a valid G .

If Q, G, V are valid, then for each (P_j, P_k) who do not have an edge and $P_j \in V$, P_j broadcasts $f_j(k)$ and P_k broadcasts $f_k(j)$ if $f_j(x)$ and $f_k(x)$ if received from the dealer at time Δ . Otherwise, they broadcast \perp . Let V' be the set of parties in V and the parties they do not have an edge to.

If G or Q or V from broadcast (continue, Q, G, V) is invalid, then wait until a broadcast (async, A, Q_a) from the dealer is received. Go to Protocol 6.6.2 on receiving (async, A, Q_a).

- (d) If \perp is received then set $b_i = 0$ and participate in an instance of Π_{BA} . If the output is 1, then go to Protocol 6.6.2. Otherwise, wait until a broadcast (async, A, Q_a) from the dealer is received. Go to Protocol 6.6.2 on receiving (async, A, Q_a).
8. (Clique Expansion or Restart (for the dealer)) At time $4T_{\text{BC}} + T_{\text{BA}}$, the dealer adds P_i in W if the broadcast of $P_i \in V'$ in the previous step is \perp or if the broadcast is not $F(i, j)$. If $|W| > 0$, then the dealer sets $U = U \cup W$ and broadcasts (restart, U). Otherwise, the dealer sets clique $Q_a = Q \cup V$ and broadcasts (sync, G, Q_a).
 9. (Local Computation: Deciding on exit route or restart (for all)) At time $5T_{\text{BC}} + T_{\text{BA}}$, every P_i does as follows:
 - (a) If (restart, U) is received, then set $b_i = 0$ and participate in an instance of Π_{BA} . If the output is 1, then go to Protocol 6.6.2. Otherwise, restart the protocol from

Step 1 with U and W reset to \emptyset .

- (b) If (sync, G, Q_a) is received from the broadcast of the dealer it constructs G_i in the same way as in Step 7c. It then updates G_i based on the broadcasts received **at time $4T_{\text{BC}} + T_{\text{BA}}$** and checks its validity as in Step 7c. Next, it checks if Q_a is a $(n - t_a)$ -size clique in G_i including parties in U . If the verification passes, then set $b_i = 1$ and $b_i = 0$ otherwise and participate in an instance of Π_{BA} . If the protocol output is 1 then go to Protocol 6.6.2. Otherwise, wait for (async, A, Q_a) from the dealer.
- (c) If \perp is received from the broadcast, then set $b_i = 0$ and participate in an instance of Π_{BA} . If the output is 1, then go to Protocol 6.6.2. Otherwise, wait until a broadcast (async, A, Q_a) from the dealer is received. Go to Protocol 6.6.2 on receiving (async, A, Q_a) .

The following steps are executed by a party when it receives an output of 1 from any Π_{BA} instance. Otherwise, parties continue to participate in Π_{WSS} iterations.

Protocol 6.6.2: $\Pi_{\text{WSS}}^{\text{Output}}$

Condition for Output: Parties output via (async, A, Q_a) only after local time $(t_s - t_a + 1) \cdot (5T_{\text{BC}} + 2T_{\text{BA}})$. Parties output via (sync, G, Q_a) only before local time $(t_s - t_a + 1) \cdot (5T_{\text{BC}} + 2T_{\text{BA}})$.

Upon receiving (sync, G, Q_a) or (async, A, Q_a) from the dealer, each P_i verifies G, Q_a or A, Q_a as follows: It constructs G_i or A_i exactly the way the dealer does in the respective steps based on the broadcasts available until now. P_i continues to update G_i or A_i based on the broadcasts it receives if the edges in G (respectively A) are not a subset of the edges in G_i (resp. A_i) or Q_a is not a $(n - t_a)$ -size clique in G_i (resp. A_i). Otherwise, it does the following:

1. If $P_i \in Q_a \setminus U$, then it sends $f_i(j)$ to every $P_j \notin Q_a \cup U$, **waits for time 3Δ** and outputs $f_i(x)$.
2. If $P_i \notin Q_a$, it **waits for 3Δ time¹** and upon receiving $t_s + t_a + 1$ points from parties in Q_a (P_i obtains points of parties in U from the dealer's broadcast) does the following:

¹In a synchronous network, if some honest party validates Q_a at time T , other honest parties may receive and validate it by time at most $T + 2\Delta$ when the dealer is corrupt. Hence, their shares may reach parties outside Q_a at time $T + 3\Delta$. Upon receiving Q_a , each party outside it thus waits for 3Δ time to ensure that it receives all the honest parties' shares before starting error correction.

- Upon receiving $t_s + t_a + 1 + x$ points, if $x \leq t_a$ then P_i tries to correct up to x errors and simultaneously detect up to $t_a - x$ errors (Corollary 6.3.3). If the decoding is successful, then P_i outputs the reconstructed polynomial.
 - Upon receiving $t_s + t_a + 1 + x$ points, if $x > t_a$ then P_i tries to correct up to t_a errors and simultaneously detect up to $x - t_a$ errors (Corollary 6.3.4). If the decoding is successful, then P_i outputs the reconstructed polynomial. Otherwise, P_i detects that the network is synchronous. It then checks the following: If (sync, G, Q_a) is received **at time $3T_{\text{BC}}$** , then it checks the validity of G_i in the same way as in Step 7c. It checks if Q_a is a $(n - t_a)$ -size clique in G_i or including parties in U . If (sync, G, Q_a) is received **at time $5T_{\text{BC}}$** , then it first updates G_i based on the broadcasts received **at time $4T_{\text{BC}}$** and checks its validity as in Step 7c. Next, it checks if Q_a is a $(n - t_a)$ -size clique in G_i including parties in U .
It outputs $f_i(x)$ if it is received from the dealer **within Δ time** from the start, Q_a does not include any P_j such that P_j 's broadcast at time $2T_{\text{BC}}$ is $f_{ji} \neq f_i(j)$ and the above verification passes. It outputs \perp otherwise.
-

Theorem 6.6.3. *Let $T_{\text{WSS}} = (t_s - t_a + 1) \cdot (5T_{\text{BC}} + 2T_{\text{BA}}) + 3\Delta$. Protocol Π_{WSS} is perfectly-secure against an adversary corrupting up to t_s parties in the synchronous network and up to t_a parties in the asynchronous network and has the following properties.*

1. *Synchronous network:*

- (a) *t_s correctness: When the dealer is honest, at time T_{WSS} , all the honest parties output $f_i(x) = F(x, i)$ corresponding to $F(x, y)$ held by the dealer.*
- (b) *t_s privacy: The view of the adversary is independent of the honest dealer's secret s .*
- (c) *t_s weak commitment: When the dealer is corrupt, either no honest party computes an output or there exists a set of at least $t_s + t_a + 1$ honest parties P_i such that each P_i outputs $f_i(x)$ where $f_i(x) = F'(x, i)$ for some (t_s, t_s) degree polynomial $F'(x, y)$. Moreover, if some honest party computes its output at $T \leq T_{\text{WSS}}$ then all honest compute their output at the same time. If some honest party computes an output at time $T > T_{\text{WSS}}$ then all the honest parties compute their output within $T + 2\Delta$.*

2. *Asynchronous network:*

- (a) *t_a correctness: When the dealer is honest, almost-surely all the honest parties output $f_i(x) = F(x, i)$ eventually where $F(x, y)$ is held by the dealer.*

- (b) t_s privacy: The view of the adversary is independent of the honest dealer's secret s .
- (c) t_a strong commitment: When the dealer is corrupt, either no honest party computes an output or almost-surely each honest party P_i outputs $f_i(x)$ eventually such that $f_i(x) = F'(x, i)$ for some (t_s, t_s) degree polynomial $F'(x, y)$.

Proof. We first prove the properties of Π_{WSS} in the synchronous network.

1. Synchronous network:

- (a) t_s correctness: Let the dealer be honest. Since the network is synchronous, we have that the adversary can corrupt up to t_s parties and the network delay is Δ . At the start of the protocol, we also have that W, U are empty. Given this, we have that within Δ time, all the parties will have their univariate polynomial shares. Further, each pair of honest parties P_i, P_j will exchange their common points on the polynomial within time 2Δ . By the liveness and validity property of broadcast in a synchronous network, we have that $(U, \{f_i(x)\}_{i \in U})$ will also be received by all the honest parties by time T_{BC} . Thus, we have that each honest party P_i will set $R_i[j] = \text{OK}$ corresponding to every honest party P_j . Moreover, P_i sets $R_i[j] = f_i(j)$ corresponding to each $P_j \in U$ such that $f_i(j) = F(i, j)$. Thus, each honest P_i has at most t_s NRs corresponding to the corrupt parties. Further, the liveness and validity properties of broadcast ensure that the honest parties' broadcast instances successfully terminate with an output by time $2T_{\text{BC}}$. Moreover, if $R_i[j] = f_{ij}$ is broadcasted by an honest P_i then it is guaranteed that $f_{ij} = F(i, j)$ indeed holds. Given that all the above conditions hold, an honest P_i is never added to W by the dealer. This implies that the dealer is bound to find a clique Q of size at least $n - t_s + |U|$ which contains all the honest parties and the parties in U . We now have the following cases to consider:

- i. **The dealer finds Q such that $|Q| \geq n - t_a$.** This implies that at time $2T_{\text{BC}}$, the dealer receives $R_i[j] = \text{OK}$ and $R_j[i] = \text{OK}$ for each $P_i, P_j \in Q$. Due to the consistency property of broadcast in the synchronous network, we have that all the honest parties will indeed see the same R_i 's at time $2T_{\text{BC}}$ as that seen by the dealer. Further, the dealer sets $Q_a = Q$ and broadcasts (sync, G, Q_a) which will be received by all the honest parties by time $3T_{\text{BC}}$. Consequently, each honest party P_i will construct the graph G_i exactly as the dealer, and hence its verification passes. Hence, all the honest parties will participate with input 1 in Π_{BA} , and due to its liveness and validity, they will receive the output as 1 by time $3T_{\text{BC}} + T_{\text{BA}}$. Further, every honest $P_i \in Q_a$ sends its share $f_i(j)$ to every

$P_j \notin Q_a$. It waits for Δ time and outputs $f_i(x)$ at time $3T_{\text{BC}} + \Delta$. Each $P_i \notin Q_a$ receives at least $|Q| - t_s \geq t_s + t_a + 1$ points from the honest parties in Q_a . We then have two cases to consider:

- If P_i receives up to t_a erroneous points from parties in Q_a , then by Corollary 6.3.3 it will recover the same polynomial after error correction as what the dealer shared and hence output the correct $f_i(x)$ at time $3T_{\text{BC}} + T_{\text{BA}} + 3\Delta$.
 - If P_i receives more than t_a erroneous points from parties in Q_a , then by Corollary 6.3.4 we have that P_i will detect this. It in turn learns that the network is indeed synchronous. Moreover, an honest P_i would have received its share $f_i(x)$ from the dealer within time Δ and sent its pairwise points $f_i(j)$ to each P_j . Let the point received by $P_j \in Q_a$ be f_{ij} . If indeed $f_j(i) \neq f_{ij}$ did not hold for P_j , then P_j would have broadcasted $R_j[i] = f_j(i)$ by time $2T_{\text{BC}}$. Given that the dealer is honest, we have that a P_j that broadcasted an incorrect value at $2T_{\text{BC}}$ would be included in W and hence $P_j \notin Q_a$ which is a contradiction. Thus, it must hold that P_j either broadcasted $R_j[i] = \text{NR}$ or $R_j[i] = F(i, j) = f_i(j)$. Thus, P_i can identify a corrupt P_j which sends an erroneous point. In this case, P_i outputs the correct polynomial received from the dealer $f_i(x)$ by time $3T_{\text{BC}} + T_{\text{BA}} + 3\Delta$.
- ii. **The dealer broadcasts** (restart, U). In this case, we have that the dealer has added at least one party in the set W and hence added at least one new party in the set U . Due to the liveness and validity properties of broadcast in the synchronous network, we have that all the honest parties will receive the dealer's broadcast by time $3T_{\text{BC}}$. Hence, all the honest parties will participate with input 0 in Π_{BA} , and due to its liveness and validity, they will receive the output as 0 by time $3T_{\text{BC}} + T_{\text{BA}}$. Subsequently, they restart the protocol successfully and in synchronization with each other. Moreover, as argued before, it is guaranteed that no honest party gets added to W or U . Thus, after at most $t_s - t_a$ restarts, U will include at least $t_s - t_a$ corrupt parties. The dealer will thus make the polynomials of $t_s - t_a$ corrupt parties public in the subsequent run of the protocol and is guaranteed to find a clique of size $(n - t_a)$ which includes the $(n - t_s)$ honest parties and the $t_s - t_a$ parties from U whose polynomials are public. Hence, in the subsequent run, the prior case is guaranteed to occur and parties will successfully output shares on the dealer's polynomial.
- iii. **The dealer broadcasts** (continue, Q, G, V). In this case, it must hold that $|Q| <$

$n - t_a$, $|U| < t_s - t_a$ and $W = \phi$. Again, by the properties of broadcast, we have that the dealer's broadcasted message will be delivered to all the honest parties by time $3T_{BC}$. Hence, all the honest parties will participate with input 0 in Π_{BA} and due to its liveness and validity, they will receive the output as 0 by time $3T_{BC} + T_{BA}$. Thus, they will proceed to check the validity of Q, G, V and identify that it is valid. It is guaranteed that each honest $P_j \in V$, has an edge with every honest P_k and the corresponding OK is received by all the honest parties, including the dealer by time $2T_{BC}$. Hence, for each (P_j, P_k) pair where $P_j \in V$ and does not have an edge with some P_k , it is guaranteed that at least one of P_j, P_k is corrupt. Further, every honest P_k such that it does not have an edge with $P_j \in V$ broadcasts the correct $f_k(j)$ and is received by all the honest parties and the dealer by time $4T_{BC}$. Hence, once again, no honest party gets added to W . We now have two cases to consider:

- The dealer broadcasts $(\text{restart}, U)$. This implies that the broadcast of some P_k or P_j such that P_k does not have an edge with $P_j \in V$ results in a \perp or results in value not equal to $F(k, j)$ at time $4T_{BC} + T_{BA}$. The dealer adds at least one party to W and hence adds at least one new party to U . By the argument above, we have that P_k or P_j added to W is guaranteed to be corrupt. The dealer then broadcasts $(\text{restart}, U)$ which is received by all the honest parties by time $5T_{BC}$. All the honest parties will participate with input 0 in Π_{BA} and due to its liveness and validity, they will receive the output as 0 by time $5T_{BC} + 2T_{BA}$. Thus, all honest parties restart the protocol in synchronization. By the same argument as earlier, upon at most $t_s - t_a$ restarts, we have that an honest dealer will conclude the protocol by finding a $|Q| \geq n - t_a$ which includes the $(n - t_s)$ honest parties and $t_s - t_a$ parties from U .
- The dealer broadcasts (sync, G, Q_a) . In this case, it must hold that for every $P_j \in V$, such that (P_j, P_k) did not have an edge at time $2T_{BC}$, both parties indeed broadcasted the correct value $F(k, j)$ by time $4T_{BC}$, thus ensuring that (P_j, P_k) are now consistent. Given that a valid Q is of size $|Q| = n - t_s + |U|$ and the dealer has additionally resolved conflicts with $(t_s - t_a) - |U|$ parties, this implies that $Q_a = Q \cup V$ is indeed of size at least $n - t_a$. Hence, the dealer's broadcast of (sync, G, Q_a) actually contains a clique of the required size and will be received by the parties within time T_{BC} . Consequently, each honest party P_i will construct the graph G_i exactly

as the dealer, hence its verification passes. All the honest parties will thus participate with input 1 in Π_{BA} and due to its liveness and validity, they will receive the output as 1 by time $5T_{BC} + T_{BA}$. The parties then compute their output similar to the first case (**The dealer finds Q such that $|Q| \geq n - t_a$**) at time $5T_{BC} + 2T_{BA} + 3\Delta$.

- (b) t_s privacy: Observe that the only step at which the dealer reveals information regarding the secret (excluding the initial step of sharing the polynomial) corresponds to the public broadcast of $f_i(x)$ for parties in U . Note that a party P_i is added to U at time $2T_{BC}$ if its broadcast corresponding to the pairwise consistency checks results in \perp , has more than t_s NRs or has an incorrect $f_i(j)$ value. Neither of these conditions holds true for an honest party; hence, an honest party does not get added to U at this time step. Further, parties also get included to U at time $4T_{BC}$. Here, a party P_i may get added to U if its broadcast corresponding to a party $P_j \in V$ results in a \perp or has an incorrect value. Given that an honest party receives an honest dealer's broadcast of $(\text{continue}, Q, G, V)$ at time $3T_{BC}$ and its own polynomial from the dealer in time Δ , it broadcasts the required (correct) values which are received by the dealer at time $4T_{BC}$. Hence, an honest party does not get added to U . Thus, we have that from the dealer's communication, an adversary can learn at most t_s univariate polynomials corresponding to the corrupt parties, thus ensuring privacy.

Further, we show that the adversary does not learn any additional information from the broadcast of honest parties. During pairwise exchange, it is ensured that the honest parties successfully send common points to each other. Hence, every honest P_i broadcasts $R_i[j] = \text{OK}$ corresponding to every honest P_j and does not reveal any information to an adversary. Further, an honest party only broadcasts points for a party in $P_j \in V$ such that (P_i, P_j) does not have an edge by time $2T_{BC}$. Given that this does not hold for any honest P_j as argued earlier, each $f_i(j)$ revealed by an honest party P_i corresponds to a corrupt P_j , thus not revealing any information to the adversary. In conclusion, the adversary cannot learn any information beyond (at most) t_s univariate polynomial shares it can obtain from (at most) t_s corrupt parties, ascertaining t_s privacy.

- (c) t_s weak commitment: If no honest party computes an output, then the weak commitment holds trivially. Hence, we consider the case when there exists some honest party P_k which computes the output at time T . We further analyze this in the following cases:

i. P_k **computes the output via obtaining** (sync, G, Q_a) **in some iteration of the protocol:** In this case, it implies that P_k obtains the output of Π_{BA} as 1 either at time $3T_{\text{BC}} + T_{\text{BA}}$ or $5T_{\text{BC}} + 2T_{\text{BA}}$. This further implies that some honest party P_h participates in Π_{BA} with input 1. If not, then the liveness and validity of Π_{BA} would ensure that parties output 0 and not compute output via (sync, G, Q_a) . Consider the case that P_h has $b_h = 1$ in Π_{BA} instance at time $3T_{\text{BC}}$. In this case, note that P_h must have verified that the dealer's graph G is indeed the same as G_h constructed using the broadcast it receives by time $2T_{\text{BC}}$. Moreover, Q_a also satisfies the requirements. By the consistency and liveness properties of broadcast in the synchronous network, we have that the output computed by all the honest parties in the broadcast instance of the dealer is the same at time $3T_{\text{BC}}$. Similarly, by the properties of broadcast, it also holds that the output of broadcast instances computed by all the honest parties at time $2T_{\text{BC}}$ is identical to that computed by P_h . Hence, it must hold that (sync, G, Q_a) is received and verified by all the parties successfully. Hence, all the parties must have set $b_i = 1$ in the instance of Π_{BA} and obtained the output 1 at time $3T_{\text{BC}} + T_{\text{BA}}$. Given that $|Q_a| \geq n - t_a$ and all the parties are consistent with each other, we have that all the honest parties in Q_a output $f_i(x)$ such that $F'(x, i) = f_i(x)$ for some (t_s, t_s) -degree bivariate polynomial F' at time $3T_{\text{BC}} + T_{\text{BA}} + 3\Delta$ in that iteration. Now consider an honest party $P_i \notin Q_a$. By time $3T_{\text{BC}} + T_{\text{BA}} + 3\Delta$, P_i is guaranteed to receive $f_j(i)$ from each honest $P_j \in Q_a$. If P_i receives at most t_a erroneous points (points not lying on $F'(x, y)$) and additionally it holds $f_i(x) = F'(x, i)$ received from the dealer at time Δ in this iteration, then by Corollary 6.3.3, P_i must have successfully reconstructed the same $f_i(x)$ from the points of parties in Q_a and set it as its output, thus ensuring the correct output. On the other hand, suppose P_i receives more than t_a erroneous points from the parties in Q_a . In this case, by Corollary 6.3.4, P_i identifies that the network is synchronous. If P_i has not received $f_i(x)$ from the dealer by time Δ then it outputs \perp . Otherwise, P_i had received its polynomial by time Δ and sent $f_i(j)$ to every P_j , an honest party $P_j \in Q_a$ for whom the value did not match would have indeed broadcasted its own value in $R_j[i]$ at time $2T_{\text{BC}}$ or its value $f_j(i)$ would already be public if $P_j \in U$. If indeed $P_j \in Q_a$ has broadcasted $R_j[i]$ at time $2T_{\text{BC}}$ or $P_j \in U$ has $f_j(i)$ which is not equal to $f_i(j)$, then P_i identifies that the dealer is corrupt since it has included $P_j \in Q_a$ while it has sent $f_i(x)$ such that $f_i(j) \neq F'(i, j)$.

Hence, P_i outputs \perp . On the other hand, if P_j has broadcasted $R_j[i] = f_i(j)$ or $R_j[i] = \text{NR}$ at time $2T_{\text{BC}}$, then P_i identifies that P_j is corrupt and has sent it an incorrect value at time $3T_{\text{BC}} + T_{\text{BA}} + 3\Delta$. In this case, P_i ignores f_{ji} sent by P_j . If P_i successfully reconstructs a polynomial after discarding these points which is equal to $f_i(x)$ received from the dealer, then it is guaranteed that the polynomial is indeed consistent with all the honest parties in Q_a . This is because P_i only discards the points of corrupt parties who behaved inconsistently at times $2T_{\text{BC}}$ and $3T_{\text{BC}} + T_{\text{BA}} + 3\Delta$. Thus, we have that an honest $P_i \notin Q_a$ indeed outputs $f_i(x) = F'(x, i)$, where $F'(x, y)$ is the (t_s, t_s) -degree bivariate polynomial defined by the honest parties in Q_a .

The other case, that P_h has $b_h = 1$ in Π_{BA} instance at time $5T_{\text{BC}} + 2T_{\text{BA}}$ follows similarly. Here, it must also hold that the output of Π_{BA} at time $3T_{\text{BC}} + T_{\text{BA}}$ was 0. Otherwise, P_h and all the honest parties would proceed as in the former case. Thus we have that P_h received (sync, G, Q_a) at time $5T_{\text{BC}} + T_{\text{BA}}$ and accepted it, then it implies that it also received a valid $(\text{continue}, Q, G, V)$ at time $3T_{\text{BC}}$ and broadcasts of parties corresponding to V at time $4T_{\text{BC}} + T_{\text{BA}}$. If not, then P_h would have either received an invalid (sync, G, Q_a) or $(\text{restart}, U)$ or \perp or an invalid G, Q, V in $(\text{continue}, Q, G, V)$. In either of these cases, P_h would not have proceeded to execute steps designated for time beyond $4T_{\text{BC}} + T_{\text{BA}}$ and not participated in Π_{BA} with input 1, which is a contradiction. Since P_h received valid broadcasts from the dealer at time $3T_{\text{BC}}$ and $5T_{\text{BC}} + T_{\text{BA}}$, by the consistency and liveness properties of broadcast in the synchronous network, we have that all the honest parties also received it at the designated time steps. Following this, the argument for t_s weak commitment follows exactly as that for the previous case, where all the honest parties either output shares on the same polynomial or \perp at time $5T_{\text{BC}} + 2T_{\text{BA}} + 3\Delta$.

- ii. **P_h computes the output via obtaining (async, A, Q_a) at time T in some iteration of the protocol:** First note that this implies that none of the $(t_s - t_a)$ iterations terminated via the (sync, G, Q_a) path for P_h . Since the decision of output computation is taken via Π_{BA} , by its liveness and consistency property, it is guaranteed that no honest party computes its output via (sync, G, Q_a) . Note that since P_h computes the output at time T , it implies that the dealer's broadcast indeed has a valid clique which was received and verified by P_h by time $T - 3\Delta$. Moreover, by the fallback consistency property of broadcast in a synchronous network, we have that all the honest parties will receive

(*async*, A, Q_a) and the AOK messages to validate its correctness within time $(T - 3\Delta) + 2\Delta = T - \Delta$. By the fact that $|Q_a| \geq n - t_a$ and includes at least $t_s + t_a + 1$ honest parties, we have that $f_i(x)$ held by each $P_i \in Q_a$ is such that $f_i(x) = F'(x, i)$ for some (t_s, t_s) -degree bivariate polynomial $F'(x, y)$. Moreover, each $P_i \in Q_a$ will compute an output by time $(T - \Delta) + 3\Delta = T + 2\Delta$. Consider an honest $P_i \notin Q_a$. As before, since the network is synchronous, it is guaranteed to receive $f_j(i)$ from each honest $P_j \in Q_a$ by time at most T . Since $P_i \notin Q_a$ waits for time at least 3Δ upon accepting (*async*, A, Q_a), it is guaranteed to receive the points of all the honest parties before proceeding for reconstruction. At this time, if it receives less than t_a erroneous points, then it successfully recovers the correct polynomial $f_i(x) = F'(x, i)$ defined by the honest parties in Q_a and computes an output at time $T + 2\Delta$. Otherwise, P_i identifies that the network is synchronous. In this case, if the dealer was honest then P_i knows that it would have terminated via (*sync*, G, Q_a). Hence, P_i identifies that the dealer is corrupt and outputs \perp . Thus, we have that all the honest parties output $f_i(x)$ such that $f_i(x) = F'(x, i)$ holds for some (t_s, t_s) degree polynomial $F'(x, y)$. Moreover, all honest parties compute their output within a delay of 2Δ from each other.

2. Asynchronous network: We now prove the properties of Π_{WSS} in the asynchronous network.

- (a) t_a correctness: Let the dealer be honest. Since the network is asynchronous, we have that the adversary can corrupt at most t_a parties. Given this and the fact that all the honest parties' messages (including the dealer's) get delivered eventually, we have that the set of all the honest parties eventually constitutes an $(n - t_a)$ sized clique. Thus we have that via the sequence of steps corresponding to an asynchronous network, the dealer will eventually broadcast (*async*, A, Q_a) which will be validated by all the honest parties. Moreover, each honest $P_i \in Q_a$ will output a correct $f_i(x)$ which it received from an honest dealer. Now consider the case of an honest party P_i outside Q_a . An honest party $P_i \notin Q_a$ will eventually receive $f_j(i)$ from every honest party $P_j \in Q_a$. Since at most t_a parties are corrupt and can send erroneous points to P_i , by Corollary 6.3.3 we have that P_i will successfully reconstruct and output a correct $f_i(x)$ consistent with the dealer's bivariate polynomial. Moreover, if some honest party actually receives (*sync*, G, Q_a) and obtains an output, by the consistency of Π_{BA} we have that some honest party participated with input 1 in the agreement protocol. Hence, all the honest parties

will eventually receive the output as 1 and compute their output correctly.

- (b) t_s privacy: As in the synchronous case, the only step at which the dealer reveals information regarding its secret beyond the sharing of polynomials is when it broadcasts $f_i(x)$ corresponding to each $P_i \in U$. Note that the dealer adds a party in U if $R_i[j] \neq F(i, j)$ or P_i 's broadcast results in NR for more than t_s parties. Given that the network is asynchronous, an honest P_i may thus get added to U . However, it is ensured that the dealer reveals $f_i(x)$ for at most $t_s - t_a$ such parties. Thus, in the worst case, we have that the adversary learns t_s such univariate polynomials $f_i(x)$ corresponding to t_a corrupt parties and additionally $t_s - t_a$ honest parties in U . Hence, the adversary can learn exactly as much information regarding the secret as in the synchronous case, thus ensuring privacy.

Further, we show that the adversary does not learn anything beyond t_s univariate polynomials, even from the broadcast of the parties. First, observe that during the pairwise exchange, if an honest party P_i does not receive $f_j(i)$ from an honest P_j then it broadcasts NR at time T_{BC} . When it eventually receives $f_j(i)$ from P_j , it is guaranteed that $f_j(i) = f_i(j)$ and hence P_i broadcasts AOK $_j$. Hence, the broadcasts corresponding to pairwise checks do not reveal any information regarding the honest parties' secrets. Next, we have that parties may reveal information regarding their polynomial if they receive (sync, Q, G, V) from the dealer. In this case again, it is possible that V contains an honest party P_i for whom all the parties P_j not having an edge with P_i reveal their common point $f_j(i)$. However, note again that $|U \cup V| \leq t_s - t_a$, and hence, at most $t_s - t_a$ honest parties' polynomials may be revealed to the adversary. By the same argument as earlier, we have that the adversary gains no information beyond t_s univariate polynomials on the dealer's polynomial, which is exactly as in the case of the synchronous network. Thus, we have that t_s privacy holds even in the asynchronous network.

- (c) t_s strong commitment: We now show that when the network is asynchronous, irrespective of the adversary's behavior, each honest party P_i will output $f_i(x)$ such that $f_i(x) = F'(x, i)$ for some (t_s, t_s) degree polynomial $F'(x, y)$.

We will first show that given two cliques, say Q_a and Q'_a , each of size at least $n - t_a$, we have that the shares held by the honest parties in Q_a as well as Q'_a are consistent with the same (t_s, t_s) degree bivariate polynomial. Given that $|Q_a| \geq n - t_a$, $|Q'_a|$ and we have a total of n parties, it must hold that $|Q_a \cap Q'_a| \geq n - 2t_a$. Moreover, we know that $n - 2t_a \geq 2t_s + 1$. Hence, it holds that $Q_a \cap Q'_a$ contains at least $t_s + 1$ honest parties who hold polynomials that define a unique (t_s, t_s) degree bivariate

polynomial, say $F(x, y)$. Let H be the set of (at least) $t_s + 1$ honest parties such that an honest $P_i \in H$ when $P_i \in Q_a \cap Q'_a$. Further, since both Q_a and Q'_a are cliques, it holds that each honest $P_j \in Q_a \cup Q'_a$ is consistent with every $P_i \in H$. Given that the degree t_s polynomial $f_j(x)$ held by P_j is consistent with (at least) $t_s + 1$ points of $F(x, j)$, it must hold that $f_j(x) = F(x, j)$ for each $P_j \in Q_a$ as well as every $P_j \in Q'_a$. This ensures that all the honest parties belonging to different cliques of size at least $n - t_a$ are guaranteed to hold polynomials consistent with a unique (t_s, t_s) degree bivariate polynomial. Given this, we now argue that our protocol ensures t_s strong commitment in an asynchronous network.

If no honest party computes an output when the dealer is corrupt, commitment holds trivially. Thus, we consider the case when some honest party P_h computes an output. Note that this implies that P_h has received either (sync, G, Q_a) and (async, A, Q_a) and verified it to compute an output. In the former case, we have that P_h received 1 as the output of Π_{BA} during some iteration of the protocol. Hence, there exists some honest party that participated in an instance of Π_{BA} with input 1. If not, then by the validity of Π_{BA} , all the honest parties would have output 0. Hence, P_h would not have computed its output via (sync, G, Q_a) which is a contradiction. By the consistency of Π_{BA} it thus holds that all the honest parties will receive 1 as the output of Π_{BA} and eventually compute their output. This is because parties in Q_a will eventually verify the clique and compute their output. For parties outside, Q_a , by Corollary 6.3.3, we have that they will be able to reconstruct the polynomial that is consistent with the honest parties in Q_a . Similarly, if P_h computes its output via (async, A, Q_a) , the same argument holds.

□

6.7 Verifiable Secret Sharing

The weak secret sharing protocol falls short of providing the properties of verifiable secret sharing. This is because, when the dealer is corrupt and the network is synchronous, it is possible that some honest parties, specifically the parties lying outside the (n, t_a) -Star, may not receive their shares. To fix this, and ensure that all or none of the honest parties receive their shares, we follow the approach of [13]. As described in [13], this results in a verifiable secret sharing protocol Π_{VSS} with two layers. The VSS protocol ensures that all the parties hold a degree- t_s Shamir-sharing of the dealer's input secret.

Here, we give a complete description of our verifiable secret sharing protocol, which al-

lows a dealer to generate a degree- t_s sharing of its input among parties, followed by its proof.

Protocol 6.7.1: Π_{VSS}

Input: The dealer holds a secret $s \in \mathbb{F}$.

Initialisation: The dealer initialises two sets W, U to ϕ . Only W is reset in every run to \emptyset .

1. (Polynomial Share Distribution) The dealer chooses a symmetric bivariate polynomial $F(x, y)$ of degree t_s in both x, y and delivers $f_i(x) = F(x, i)$ to P_i . If $|U| > t_s - t_a$, then assign U to be the set of first $t_s - t_a$ parties lexicographically. The dealer broadcasts $(U, \{f_i(x)\}_{i \in U})$.
2. (Pair-wise exchange) **At time T_{BC}** , if $f_i(x)$ is received **at time Δ** then every P_i participates in an instance of Π_{WSS} as the dealer, say $\Pi_{\text{WSS}}^{(i)}$ with input $f_i(x)$. P_i also participates in $\Pi_{\text{WSS}}^{(j)}$ instances for every $j \in \{1, \dots, n\} \setminus U$.
3. (Pair-wise Consistency Check) **At time $T_{\text{BC}} + T_{\text{WSS}}$** P_i prepares a vector R_i of length n as follows and broadcasts it. It sets $R_i[j] = \text{NR}$ for all j if either of the following happens:
 - (a) it receives no $f_i(x)$
 - (b) the dealer's broadcast results in \perp
 - (c) some $f_j(x)$ in the broadcast $(U, \{f_i(x)\}_{i \in U})$ is of degree more than t_s
 - (d) there are indices j, k such that $f_j(k) \neq f_k(j)$ in the broadcast $(U, \{f_i(x)\}_{i \in U})$
Otherwise, it sets R_i as follows. (1) if $P_j \in U$, then $R_i[j] = f_i(j)$ (2) if $P_j \notin U$, then set (a) $R_i[j] = \text{NR}$ if f_{ji} is not computed as output in P_j 's instance $\Pi_{\text{WSS}}^{(j)}$, (b) $R_i[j] = f_i(j)$ if f_{ji} is received as output from $\Pi_{\text{WSS}}^{(j)}$ and $f_i(j) \neq f_{ji}$, (c) $R_i[j] = \text{OK}$ otherwise.
4. (Asynchronous Pair-wise Consistency Checking) The parties execute the following steps as and when they receive the required values. On receiving the broadcast $(U, \{f_i(x)\}_{i \in U})$ and polynomial $f_i(x)$ from the dealer, every $P_i \notin U$ participates in an instance of Π_{WSS} as the dealer, say $\Pi_{\text{WSS}}^{(i)}$ with input $f_i(x)$. P_i also participates in $\Pi_{\text{WSS}}^{(j)}$ instances for every $j \in \{1, \dots, n\}$. P_i broadcasts AOK_j if (a) f_{ji} is computed from $\Pi_{\text{WSS}}^{(j)}$ from $P_j \notin U$ and $f_i(j) = f_{ji}$ (b) $f_j(i)$ for $P_j \in U$ satisfies $f_i(j) = f_j(i)$.
5. (Restart or Clique Finding) **At time $2T_{\text{BC}} + T_{\text{WSS}}$** , the dealer puts $P_i \notin U$ in W if either happens (a) P_i 's broadcast of R_i resulted in \perp or (b) P_i 's broadcasted R_i has more than t_s NRs or (c) $R_i[j] \neq F(i, j)$ when $R_i[j] \neq \text{OK}$ and $R_i[j] \neq \text{NR}$.

The dealer makes a graph G with n vertices corresponding to n parties. There is an edge when $R_i[j] = R_j[i] = \text{OK}$. There is no edge if $R_i[j] = \text{NR}$ or $R_j[i] = \text{NR}$. The dealer finds a clique Q of size $n - t_s + |U|$ in the graph including U . If $|Q| \geq n - t_a$, then the dealer sets $Q_a = Q$ and broadcasts (sync, G, Q_a) . Otherwise, if $|W| > 0$, then the dealer

sets $U = U \cup W$ and broadcasts (restart, U). Otherwise, it broadcasts (continue, Q, G, V), where V is a set of $(t_s - t_a) - |U|$ parties (vertices) outside $Q \cup U$.

6. (Asynchronous Clique Finding) The dealer executes the following steps as and when it receives the required messages. First, the dealer initiates a graph A with parties as vertices with edges between a pair of parties in U . On receiving broadcasts AOK_{ij} and AOK_{ji} from $P_i, P_j \notin U$, it adds an edge between P_i, P_j . On receiving broadcast AOK_{ij} from $P_i \notin U, P_j \in U$, it adds an edge between P_i, P_j . Each time there is an update in A , it invokes $(C, D, E, F) \leftarrow \text{Star}(A)$ (Protocol 6.4.2) If $|F| > n - t_a$, it sets $Q_a = F$ and broadcasts (async, A, Q_a).
7. (Conflict Resolution for Clique Expansion or Restart) At time $3T_{\text{BC}} + T_{\text{WSS}}$, the parties do the following:

- (a) If (sync, G, Q_a) is received, then P_i verifies G, Q_a as follows. It checks the validity of G_i in the same way as in Step 7c. It checks if Q_a is a $(n - t_a)$ -size clique in G_i including parties in U . If the verification passes, then set $b_i = 1$ and $b_i = 0$ otherwise and participate in an instance of Π_{BA} . If the protocol output is 1 then go to Protocol 6.7.2. Otherwise, wait for (async, A, Q_a) from the dealer.
- (b) If (restart, U) is received, then set $b_i = 0$ and participate in an instance of Π_{BA} . If the output is 1, then go to Protocol 6.7.2. Otherwise, restart the protocol from Step 1.
- (c) If (continue, Q, G, V) is received, then set $b_i = 0$ and participate in an instance of Π_{BA} . If the output is 1, then go to Protocol 6.7.2. Otherwise, when the output is 0, verify Q, G, V . For this, construct G_i exactly as the dealer did based on the broadcasts available at time $2T_{\text{BC}} + T_{\text{WSS}}$ at Step 5. G is marked as invalid if
 - i. it is different from G_i AND
 - ii. there is a pair $P_j, P_k \notin U$ such that $R_j[k] \neq R_k[j]$ or there is a pair $P_j \notin U, P_k \in U$ such that $R_j[k] \neq f_k(j)$.

Q is invalid if it is not a clique in a valid G of size at least $n - t_a$ and does not include parties in U . V is invalid if it is not a set of $(t_s - t_a) - |U|$ parties (vertices) outside $Q \cup U$ in a valid G .

If Q, G, V are valid, then for each (P_j, P_k) who do not have an edge and $P_j \in V$, P_j broadcasts $f_j(k)$ and P_k broadcasts $f_k(j)$ if $f_j(x)$ and $f_k(x)$ if received from the dealer at time Δ . Otherwise, they broadcast \perp . Let V' be the set of parties in V and the parties they do not have an edge to.

- If G or Q or V from broadcast (continue, Q, G, V) is invalid, then wait until a broadcast (async, A, Q_a) from the dealer is received. Go to Protocol 6.7.2 on receiving (async, A, Q_a).
- (d) If \perp is received then set $b_i = 0$ and participate in an instance of Π_{BA} . If the output is 1, then go to Protocol 6.7.2. Otherwise, wait until a broadcast (async, A, Q_a) from the dealer is received. Go to Protocol 6.7.2 on receiving (async, A, Q_a).
8. (Clique Expansion or Restart (for the dealer)) **At time $4T_{BC} + T_{WSS} + T_{BA}$** , the dealer adds P_i in W if the broadcast of $P_i \in V'$ in the previous step is \perp or if the broadcast is not $F(i, j)$. If $|W| > 0$, then the dealer sets $U = U \cup W$ and broadcasts (restart, U). Otherwise, the dealer sets clique $Q_a = Q \cup V$ and broadcasts (sync, G, Q_a).
9. (Local Computation: Deciding on exit route or restart (for all)) **At time $5T_{BC} + T_{WSS} + T_{BA}$** , every P_i does as follows:
- (a) If (restart, U) is received, then set $b_i = 0$ and participate in an instance of Π_{BA} . If the output is 1, then go to Protocol 6.7.2. Otherwise, restart the protocol from Step 1 with U and W reset to \emptyset .
- (b) If (sync, G, Q_a) is received from the broadcast of the dealer it constructs G_i in the same way as in Step 7c. It then updates G_i based on the broadcasts received **at time $4T_{BC} + T_{WSS} + T_{BA}$** and checks its validity as in Step 7c. Next, it checks if Q_a is a $(n - t_a)$ -size clique in G_i including parties in U . If the verification passes, then set $b_i = 1$ and $b_i = 0$ otherwise and participate in an instance of Π_{BA} . If the output of the protocol is 1 then go to Protocol 6.7.2. Otherwise, wait for (async, A, Q_a) from the dealer.
- (c) If \perp is received from the broadcast, then set $b_i = 0$ and participate in an instance of Π_{BA} . If the output is 1, then go to Protocol 6.7.2. Otherwise, wait until a broadcast (async, A, Q_a) from the dealer is received. Go to Protocol 6.7.2 on receiving (async, A, Q_a).

Protocol 6.7.2: $\Pi_{VSS}^{\text{Output}}$

Condition for Output: Parties output via (async, A, Q_a) only after local time $(t_s - t_a + 1) \cdot (5T_{BC} + T_{WSS} + 2T_{BA})$. Parties output via (sync, G, Q_a) only before local time $(t_s - t_a + 1) \cdot (5T_{BC} + T_{WSS} + 2T_{BA})$.

Upon receiving (sync, G, Q_a) or (async, A, Q_a) from the dealer, each P_i verifies G, Q_a or

A, Q_a as follows: It constructs G_i or A_i exactly the way the dealer does in the respective steps based on the broadcasts available until now. P_i continues to update G_i or A_i based on the broadcasts it receives if the edges in G (respectively A) are not a subset of the edges in G_i (resp. A_i) or Q_a is not a $(n - t_a)$ -size clique in G_i (resp. A_i). Otherwise, it does the following:

1. If $P_i \in Q_a$, outputs $f_i(x)$.
 2. If $P_i \notin Q_a$ then upon computing f_{j_i} as output from $\Pi_{\text{WSS}}^{(j)}$ corresponding to $t_s + 1$ parties $P_j \in Q_a$, P_i reconstructs its polynomial $f_i(x)$ and outputs it.
-

Theorem 6.7.3. *Let $T_{\text{VSS}} = (t_s - t_a + 1) \cdot (5T_{\text{BC}} + T_{\text{WSS}} + 2T_{\text{BA}})$. Protocol Π_{VSS} is perfectly-secure against an adversary corrupting up to t_s parties in the synchronous network and up to t_a parties in the asynchronous network and has the following properties.*

1. *Synchronous network:*

- (a) t_s correctness: *When the dealer is honest, at time T_{VSS} , all the honest parties output $s_i = f_i(0)$.*
- (b) t_s privacy: *The view of the adversary is independent of the honest dealer's secret s .*
- (c) t_s strong commitment: *When the dealer is corrupt, either no honest party computes an output or each honest party P_i is such that it outputs s_i . Moreover, it holds that $s_i = f'(i)$ for some degree- t_s polynomial $f'(x)$. Also, if some honest party outputs by time T , then all honest parties have an output by time $T + 2\Delta$.*

2. *Asynchronous network:*

- (a) t_a correctness: *When the dealer is honest, almost-surely all the honest parties output $s_i = f_i(0)$ eventually.*
- (b) t_s privacy: *The view of the adversary is independent of the honest dealer's secret s .*
- (c) t_a strong commitment: *When the dealer is corrupt, either no honest party computes an output or almost-surely each honest party P_i outputs s_i eventually such that $s_i = f'(i)$ for some degree- t_s polynomial $f'(x)$.*

Proof. At a very high level, the proof follows closely to that of Π_{WSS} . We first prove the properties of Π_{VSS} in the synchronous network.

1. *Synchronous Network:*

(a) t_s correctness: Consider the dealer to be honest. Given that the network is synchronous, we have that the network has a delay of at most Δ . Thus, each honest party P_i will receive its $f_i(x)$ from the dealer within time Δ . Moreover, the dealer's broadcast of $(U, \{f_i(x)\}_{i \in U})$ will be received within time T_{BC} from the start and initiate the instance of $\Pi_{WSS}^{(i)}$. Further, every honest party will also participate in the instances initiated by all the other honest parties. We have that by time $T_{BC} + T_{WSS}$, all the honest parties will compute output in the Π_{WSS} instance of every other honest party. Thus, we have that by time $T_{BC} + T_{WSS}$, P_i has all the required information to compute the vector R_i . Hence, it will compute R_i such that $R_i[j] = \text{OK}$ for every honest P_j and the correct $f_i(j)$ corresponding to every $P_j \in U$ and broadcasts it. By the liveness and validity property of broadcast in the synchronous network, we have that all the honest parties broadcast will be received successfully by time $2T_{BC} + T_{WSS}$. Hence, we have that no honest party gets added to W . Similar to Π_{WSS} , we now have the following cases to consider:

- i. **The dealer finds a Q such that $|Q| \geq n - t_a$.** This implies that the dealer received the broadcasts of all the parties in Q by time $2T_{BC} + T_{WSS}$. By the consistency property of broadcast, we have that all the honest parties would also have received the same. Further, we have that the dealer will broadcast (sync, G, Q_a) which will be received by all the parties at time $3T_{BC} + T_{WSS}$. And hence, all the honest parties will participate in Π_{BA} with input 1. By the liveness and validity of Π_{BA} in the synchronous network, all the honest parties will output 1 at time $3T_{BC} + T_{WSS} + T_{BA}$ and hence compute the output as follows. Every $P_i \in Q_a$ will output the polynomial $f_i(x)$ it received from the dealer. Now consider an honest party $P_i \notin Q_a$. Since $|Q_a| \geq n - t_a$, we have that at least $n - t_a - t_s \geq t_s + 1$ honest parties. Thus, it holds that P_i must have computed f_{ji} as the output in P_j 's instance of $\Pi_{WSS}^{(j)}$. Hence, P_i has at least $t_s + 1$ points on a t_s degree polynomial. We now consider the case when P_i has computed its output in $\Pi_{WSS}^{(j)}$ for some corrupt party $P_j \in Q_a$. Since (sync, G, Q_a) is such that the dealer honestly computed Q_a , it must hold that Q_a was indeed a clique at time $2T_{BC} + T_{WSS}$. This implies that each honest $P_k \in Q_a$ is broadcasted $R_k[j] = \text{OK}$ corresponding to every corrupt $P_j \in Q_a$. By the t_s weak commitment property of Π_{WSS} , we have that all the honest parties would have indeed computed f_{kj} which lie on a unique t_s degree polynomial. Given that at least $t_s + 1$ honest parties broadcasted OK to such a corrupt party P_j , it must indeed hold that P_j participated with the dealer's correct polynomial

$f_j(x)$ in $\Pi_{\text{WSS}}^{(j)}$. Hence, we have that even if $P_i \notin Q_a$ has computed an output f_{ji} in $\Pi_{\text{WSS}}^{(j)}$ corresponding to a corrupt P_j , it must hold that $f_{ji} = f_i(j)$. Hence, the polynomial interpolated by $P_i \notin Q_a$ is indeed $f_i(x) = F(x, i)$ where $F(x, y)$ is the (t_s, t_s) degree bivariate polynomial held by the dealer.

ii. **The dealer broadcasts** (restart, U). In this case, we have that the dealer added at least one party to W , and hence added at least one new party to U . Further, by the validity of broadcast, all the honest parties will receive (restart, U) at time $3T_{\text{BC}} + T_{\text{WSS}}$ and set their input to Π_{BA} as 0. By the validity of Π_{BA} , all parties will output 0 at time $3T_{\text{BC}} + T_{\text{WSS}} + T_{\text{BA}}$ and consequently, restart the protocol in synchronization. Moreover, note that a party is added to W if and only if it broadcasts an incorrect value or its broadcast results in more than t_s NRs. However, given that each honest P_i computes f_{ji} in $\Pi_{\text{WSS}}^{(j)}$ corresponding to every honest party P_j , and it receives $f_i(x)$ from the dealer within Δ time, we have that $R_i[j] = \text{OK}$ for every honest P_j . Moreover, for every corrupt P_j such that $f_{ji} \neq f_i(j)$, it holds that $R_i[j] = f_i(j)$ broadcasted by P_i is indeed the correct value. Hence, an honest party is never added to W and hence not added to U . Given this observation, we have that upon $t_s - t_a$ restarts of the protocol, an honest dealer would have added $t_s - t_a$ corrupt parties to U , and hence their polynomials would be public. Thus, in the subsequent iteration, the dealer is bound to find a clique of size $n - t_a$ and successfully terminate via the former path of (sync, G, Q_a).

iii. **The dealer broadcasts** (continue, Q, G, V). This implies that $|Q_a| < n - t_a$, $|U| < t_s - t_a$ and $W = \phi$. Since the dealer broadcasts this at time $2T_{\text{BC}} + T_{\text{WSS}}$, we have that all the honest parties receive it by $3T_{\text{BC}} + T_{\text{WSS}}$ and participate with input 0 in Π_{BA} . Parties will thus output 0 at time $3T_{\text{BC}} + T_{\text{WSS}} + T_{\text{BA}}$ and proceed to verify the dealer's broadcasted sets. By the validity of broadcast at time $2T_{\text{BC}} + T_{\text{WSS}}$, we have that all the honest parties will identify Q, G, V to be valid. Moreover, we have that an honest $P_i \in V$, would have computed an output in $\Pi_{\text{WSS}}^{(j)}$ every honest P_j and hence neither P_i nor P_j broadcast their value at this stage. Also, each honest P_i broadcasts the correct $f_i(j)$ for every corrupt $P_j \in V$ which is received by all the honest parties including the dealer at time $4T_{\text{BC}} + T_{\text{WSS}} + T_{\text{BA}}$. Hence, an honest party does not get added to W . We now have two cases to consider:

- The dealer broadcasts (restart, U). This implies that the broadcast of some corrupt party P_j either resulted in a \perp or had an incorrect value. In either

case, the dealer adds this party to W and thus has identified a new party to be added to U . The dealer's broadcast of $(\text{restart}, U)$ is received by all the honest parties by time $5T_{\text{BC}} + T_{\text{WSS}} + T_{\text{BA}}$ who participated with input 0 in an instance of Π_{BA} . Thus, by the validity of Π_{BA} , the honest parties will obtain 0 as the output at time $5T_{\text{BC}} + T_{\text{WSS}} + 2T_{\text{BA}}$ and restart the protocol in synchronization.

- The dealer broadcasts (sync, G, Q_a) . This implies that for every $P_j \in V$ such that (P_j, P_k) did not have an edge, both P_j and P_k broadcasted the correct $f_j(k)$ by time $4T_{\text{BC}} + T_{\text{WSS}} + T_{\text{BA}}$. By the validity of broadcast, we have that all the honest parties indeed have the same output. Due to this, parties will participate in Π_{BA} with input 1. By the validity of Π_{BA} , we have that all the honest parties will output 1 at time $5T_{\text{BC}} + T_{\text{WSS}} + 2T_{\text{BA}}$ and compute their output. The output computation will be successful due to the same argument as the first case (**The dealer finds a Q such that $|Q| \geq n - t_a$.**) and hence we avoid repetition.

In all the above cases, note that parties compute their output within time T_{VSS} .

- (b) t_s privacy: Apart from sending the pairwise shares to each party, the dealer reveals information corresponding to its secret only when it broadcasts $f_i(x)$ corresponding to every $P_i \in U$. Moreover, a party P_i is added to U only if it broadcasts the incorrect value corresponding to $f_i(j)$ or its broadcasts result in a \perp or more than t_s NRs. Given this, we note that no honest party gets added to U . Thus, we have that every party in U is corrupt when the dealer is honest and hence already knows the $f_i(x)$ broadcasted by the dealer. Now consider the values broadcasted by the honest parties. Since every honest P_i computes an output in $\Pi_{\text{WSS}}^{(j)}$ instance of every honest P_j , we have that by time $2T_{\text{BC}} + T_{\text{WSS}}$, P_i broadcasts their $R_i[j] = \text{OK}$. By the validity property of broadcast, this will be received by all the honest parties including the dealer and the consistency graph constructed by all the honest parties contains an edge for every honest (P_i, P_j) . The only other time step at which an honest party P_i broadcasts $f_i(j)$ for some party P_j is when the dealer broadcasts $(\text{continue}, Q, G, V)$. Again, at this step, an honest $P_i \in V$ will only broadcast the correct $f_i(j)$ corresponding to every corrupt P_j . And similarly, an honest $P_i \notin V$ will broadcast $f_i(j)$ for every corrupt $P_j \in V$. These are the values that the adversary already knows having obtained the $f_j(x)$ corresponding to every P_j and hence does not learn anything additionally. Finally, we have that for every corrupt P_j , the adversary learns $f_i(j)$ corresponding to an honest P_i in its instance of

$\Pi_{\text{WSS}}^{(i)}$. However, this information is already available to the adversary due to its univariate polynomial share $f_j(x)$. Further, the t_s privacy of Π_{WSS} ensures that the adversary's view remains independent of an honest party's polynomial $f_i(x)$. In conclusion, we have that the adversary can learn at most t_s univariate polynomials $f_j(x)$ corresponding to (at most) t_s corrupt parties, thus ensuring t_s privacy.

(c) t_s strong commitment: Consider the case when the dealer is corrupt. If no honest party computes an output, then strong commitment holds trivially. We thus consider the case when some honest party, say P_k , computes its output. We now have two cases to consider:

- i. P_k **computes the output by obtaining** (sync, G, Q_a) **in some iteration of the protocol.** This implies that P_k received 1 as the output of Π_{BA} in some iteration of the protocol before T_{VSS} . This further implies that there exists some honest party P_h which participated in Π_{BA} with input 1. If not, then all the honest parties would have set their input as 0, and by the validity property of Π_{BA} , all the parties would have received 0. In this case, parties would not have output via (sync, G, Q_a) which is a contradiction. Thus, it must be that some P_h set $b_h = 1$ as its input to Π_{BA} . This also implies that P_h received the dealer's broadcasts as well as the necessary broadcasts from the parties as per the synchronous time steps and verified it. By the liveness and consistency properties of broadcast, we thus have that all the honest parties must have computed the same output in all the broadcast instances and set their input to Π_{BA} as 1. This in turn implies that all the honest parties will compute their output via (sync, G, Q_a) . Further, since accepting (sync, G, Q_a) involves verifying the dealer's graph based on the broadcast of parties at time $2T_{\text{BC}} + T_{\text{WSS}}$, it must hold that honest parties indeed broadcasted their R_i vector at time $T_{\text{BC}} + T_{\text{WSS}}$. This also implies that there exist honest parties obtained the output of $\Pi_{\text{WSS}}^{(j)}$ instantiated by some corrupt party $P_j \in Q_a$ within time T_{WSS} of its start. By the t_s weak commitment property of Π_{WSS} , it must thus hold that all the honest parties that compute an output in $\Pi_{\text{WSS}}^{(j)}$ do so within the same time and hence have their output by time $T_{\text{BC}} + T_{\text{WSS}}$. Now consider the honest parties in Q_a . Since parties verify the validity of the clique Q_a , it is ensured that all the honest parties in Q_a are actually consistent with each other. Thus, each honest $P_i \in Q_a$ must hold $f_i(x)$ such that $f_i(x) = F'(x, i)$ for some (t_s, t_s) degree bivariate polynomial $F'(x, y)$. Now consider an honest party $P_i \notin Q_a$. Given that $|Q_a| \geq n - t_a$, we have that there are at least $n - t_a - t_s \geq t_s + 1$ honest

parties in Q_a . Hence, it is guaranteed that an honest $P_i \notin Q_a$ will compute an output in Π_{WSS} instances of at least $t_s + 1$ parties from Q_a . Consequently, P_i can reconstruct its $f_i(x)$ consistent with the polynomial $F'(x, y)$ defined by the shares of the honest parties in Q_a . Finally, in case P_i has computed its output in $\Pi_{\text{WSS}}^{(j)}$ for some corrupt party $P_j \in Q_a$, then by the same argument as in the case of t_s correctness, we have that the output f_{ji} computed by P_i is indeed the same as $F'(i, j)$. This holds since the corrupt $P_j \in Q_a$ is consistent with at least $t_s + 1$ honest parties, thus ensuring that $f_j(x)$ shared by P_j in its Π_{WSS} instance is actually $F'(x, j)$. Hence, we have that an honest $P_i \notin Q_a$ successfully reconstructs its $f_i(x) = F'(x, i)$ within time T_{VSS} ensuring t_s strong commitment.

- ii. **P_k computes its output via obtaining (async, A, Q_a) at time T in some iteration of the protocol.** We first note that in this case, $T > T_{\text{VSS}}$ since the parties did not output via (sync, G, Q_a) in any of the $(t_s - t_a)$ iterations of the protocol. Since P_k computes its output at time T , it implies that it received (async, A, Q_a) and verified the broadcasts of all the parties in Q_a by time T . Given that the network is synchronous, by the t_s fallback consistency property of broadcast, we have that all the honest parties will receive (async, A, Q_a) as well as the corresponding broadcasts by time at most $T + 2\Delta$. Thus, an honest party $P_i \in Q_a$ will output $f_i(x)$ by time $T + 2\Delta$. Since $|Q_a| - t_s \geq t_s + 1$, we have that the univariate polynomial shares of all the honest parties in Q_a indeed define a (t_s, t_s) degree bivariate polynomial $F'(x, y)$ such that $f_i(x) = F'(x, i)$ holds for each $P_i \in Q_a$. Further, since Q_a is verified to be a clique by some honest party at time T , it implies that $\Pi_{\text{WSS}}^{(i)}$ instance of each honest $P_i \in Q_a$ terminated before time T . By the t_s correctness property of Π_{WSS} in the synchronous network, we have that all the honest parties compute their output at the same time and hence would have computed their output in $\Pi_{\text{WSS}}^{(i)}$ before time T . This further implies that every honest $P_j \notin Q_a$ must have computed its output in at least $t_s + 1$ instances of Π_{WSS} corresponding to the honest parties in the clique, and hence will compute its output by time $T + 2\Delta$ in the worst case upon receiving (async, A, Q_a) and validating it. Moreover, the correctness of the polynomial $f_j(x)$ interpolated by an honest $P_j \notin Q_a$ can be established as in the earlier cases. We avoid repeating the argument since it's identical to the prior cases.

2. **Asynchronous Network:** We now prove the properties of Π_{VSS} in the asynchronous network.

- (a) t_a correctness: Let the dealer be honest. Given that the network is asynchronous, we have that the adversary can corrupt at most t_a of the parties. Given this, we have that eventually, each honest P_i will successfully compute the output in $\Pi_{\text{WSS}}^{(j)}$ corresponding to every honest P_j and broadcast AOK_j . Thus, it is guaranteed that the dealer will eventually identify a clique Q_a of size at least $n - t_a$ consisting of all the honest parties and broadcast it. Thus, if the parties do not compute their output via (sync, G, Q_a) , then we have that they will eventually receive (async, A, Q_a) and compute their output. For every honest $P_i \in Q_a$, we have that it will output $f_i(x)$ received from the dealer. On the other hand, an honest $P_i \notin Q_a$ will eventually compute its output in $\Pi_{\text{WSS}}^{(j)}$ corresponding to at least $t_s + 1$ honest parties in Q_a and hence compute its output as in the prior cases. If $P_i \notin Q_a$ computes f_{ji} as output in $\Pi_{\text{WSS}}^{(j)}$ corresponding to some corrupt $P_j \in Q_a$, then by the same argument as the synchronous case, $f_{ji} = f_i(j)$ must hold where $f_i(x) = F(x, i)$ corresponding to the (t_s, t_s) degree bivariate polynomial held by the dealer.
- In the case that some honest party computes its output via (sync, G, Q_a) , then it must hold that some honest party participated with input 1 in Π_{BA} instance. Otherwise, all the honest parties would have input 0 and the validity of Π_{BA} would ensure that parties received 0 and do not compute their output via (sync, G, Q_a) which is a contradiction. Thus, we have that some honest party input 1 to Π_{BA} . By the consistency of Π_{BA} , we first have that all the honest parties will output 1 and compute their output via (sync, G, Q_a) . Moreover, the party which participated with 1 would have verified the validity of Q_a before accepting it. Thus, all the honest parties will eventually validate Q_a , accept it and compute their output as described in the prior cases.
- (b) t_s privacy: The argument for t_s privacy is exactly as in the case of the synchronous network, hence we avoid repetition.
- (c) t_a strong commitment: Let the dealer be corrupt. Since the network is asynchronous, we have that the dealer can corrupt at most t_a parties. Note that t_s strong commitment was already achieved by the weaker variant of Π_{WSS} . This property follows very closely to Π_{VSS} . Strong commitment holds trivially if no honest party computes an output in a corrupt dealer's instance. Thus, we consider the case when some honest party P_h computes an output. We have two cases here: either P_h computes an output via (sync, G, Q_a) or (async, A, Q_a) . In the former case, we have that some honest party participated in an instance of Π_{BA} with input 1. If not then the validity of Π_{BA} would ensure that parties output 0 and do not compute

their output via (sync, G, Q_a) which is a contradiction. Thus, we have that there exists some honest party which input 1 to Π_{BA} . This honest party is guaranteed to have checked the validity of Q_a as required in the protocol at designated time steps. Hence, it must hold that Q_a is indeed a clique, which will eventually be verified by all the honest parties to compute the output. Every $P_i \in Q_a$ will thus output $f_i(x)$ such that $f_i(x) = F'(x, i)$ for some (t_s, t_s) -degree bivariate polynomial defined by the honest parties in Q_a . Further, given that $|Q_a| \geq n - t_a$, we have that the number parties in $Q_a \setminus U$ is at least $n - t_a - (t_s - t_a)$, that is $n - t_s$. Of these, we are guaranteed to have at least $n - 2t_s \geq t_s + 1$ honest parties. For every $P_i \notin Q_a$, it is thus ensured that P_i will compute its output f_{ji} in the instance $\Pi_{\text{WSS}}^{(j)}$ of every honest $P_j \in Q_a$, and hence successfully reconstruct $f_i(x) = F'(x, i)$ eventually. In the latter case, it must hold that P_h did not output via (sync, G, Q_a) in any of the $(t_s - t_a)$ iterations of the protocol. Since P_h indeed computes its output upon receiving (async, A, Q_a) , it must hold that P_h verified the validity of Q_a . This implies that all the honest parties will eventually receive the same and compute their output. As in Π_{WSS} , we also have that the shares of the honest parties in two different cliques Q_a and Q'_a define the same (t_s, t_s) -degree bivariate polynomial. Hence, irrespective of which $n - t_a$ sized clique an honest party accepts, it is ensured that its output will be consistent with all the honest parties. □

6.8 Verifiable Triple Sharing

In this section, we give our triple sharing protocol which was discussed in Section 6.2.2.

Protocol 6.8.1: Π_{VTS}

Input: The dealer holds $2t_s + 1$ random multiplication triples denoted by $\{(a_i, b_i, c_i)\}_{i \in \{1, \dots, 2t_s + 1\}}$.

Common Input: $n + 1$ distinct elements from \mathbb{F} , $1, \dots, n$ and β .

Condition: Parties continue to resolve conflicts by publicly reconstructing $X(i), Y(i), Z(i)$ for $\text{NOK}(i)$ received from a party P_i until they discard the dealer or compute an output.

1. The dealer generates the degree- t_s sharings by executing Π_{VSS} to compute $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$ for every $i \in \{1, \dots, 2t_s + 1\}$.
2. Upon computing the output in all the instances of Π_{VSS} , **wait for the time to be a multiple of Δ** . Then, for each $i \in \{1, \dots, t_s + 1\}$, parties locally set $\langle x_i \rangle = \langle a_i \rangle$, $\langle y_i \rangle = \langle b_i \rangle$ and $\langle z_i \rangle = \langle c_i \rangle$.

3. Let $X(\cdot)$ and $Y(\cdot)$ be the unique polynomials of degree at most t_s defined by the points $\{(i, x_i)\}_{i \in \{1, \dots, t_s+1\}}$ and $\{(i, y_i)\}_{i \in \{1, \dots, t_s+1\}}$ respectively. The parties locally compute $\langle x_i \rangle = \langle X(i) \rangle$ and $\langle y_i \rangle = \langle Y(i) \rangle$, for each $i \in \{t_s + 2, \dots, 2t_s + 1\}$.¹
4. Parties invoke Π_{Beaver} with $\{\langle x_i \rangle, \langle y_i \rangle, \langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle\}_{i \in \{t_s+2, \dots, 2t_s+1\}}$ and **wait for time T_{Beaver}** . Upon obtaining the output $\{\langle z_i \rangle\}_{i \in \{t_s+2, \dots, 2t_s+1\}}$ where $z_i = x_i y_i$ for every $i \in \{t_s + 2, \dots, 2t_s + 1\}$, **wait for the time to be a multiple of Δ** and then proceed to the next step.
5. Let $Z(\cdot)$ be the polynomial of degree at most $2t_s$ defined by the points $\{(i, z_i)\}_{i \in \{1, \dots, 2t_s+1\}}$.
6. Parties compute $\{(\langle X(i) \rangle, \langle Y(i) \rangle, \langle Z(i) \rangle)\}$ for each $i \in \{2t_s + 2, \dots, n\}$ using $\{(\langle X(i) \rangle, \langle Y(i) \rangle, \langle Z(i) \rangle)\}_{i \in \{1, \dots, 2t_s+1\}}$.
7. For each $P_i \in \mathcal{P}$, parties invoke Π_{privRec} 3 times with $\langle X(i) \rangle$, $\langle Y(i) \rangle$ and $\langle Z(i) \rangle$ as input respectively to enable P_i to privately reconstruct $X(i)$, $Y(i)$ and $Z(i)$. Each party **waits for time T_{PrivRec}** . Upon computing the output, **wait for the time to be a multiple of Δ** and proceed to the next step.
8. If $X(i) \cdot Y(i) = Z(i)$ holds, P_i broadcasts $\text{OK}(i)$, and broadcasts $\text{NOK}(i)$ otherwise. Parties publicly reconstruct $X(i)$, $Y(i)$, $Z(i)$ for each $\text{NOK}(i)$ by broadcasting their shares. Each party **waits for T_{BC}** before proceeding to the next step.
9. The dealer constructs a set $\text{OK} = \{i | \text{OK}(i) \text{ was received from } P_i\}$. Once $|\text{OK}| \geq n - t_s$, the dealer constructs a set NOK of size $(n - t_a) - |\text{OK}|$ such that $\text{NOK} \subset \mathcal{P} \setminus \text{OK}$ and broadcasts (OK, NOK) . Parties **wait for time T_{BC}** before proceeding.
10. Parties publicly reconstruct $X(i)$, $Y(i)$, $Z(i)$ for each $i \in \text{NOK}$, by broadcasting their shares. **Wait for time T_{BC}** . Upon receiving $X(i)$, $Y(i)$, $Z(i)$, verify that $X(i) \cdot Y(i) = Z(i)$ holds. If not, then discard the dealer.
11. Upon receiving $\text{OK}(i)$ from each $i \in \text{OK}$, completing the prior check for each $i \in \text{NOK}$, and ensuring that $\text{OK} \cup \text{NOK} \geq n - t_a$, each party proceeds to the next step.
12. Discard the dealer if $X(i) \cdot Y(i) = Z(i)$ does not hold for some party which broadcasted $\text{NOK}(i)$. If the dealer is discarded, parties output a default degree- t_s sharing of a publicly known value. Otherwise, parties locally compute and output their shares of $(\langle X(\beta) \rangle, \langle Y(\beta) \rangle, \langle Z(\beta) \rangle)$, where $\beta \neq i$ for every $i \in \{1, \dots, n\}$.

¹Computing a new point on a polynomial of degree t_s is a linear function of $t_s + 1$ given unique points on the same polynomial.

Theorem 6.8.2. *Protocol Π_{VTS} is perfectly-secure against an adversary corrupting up to t_s parties in the synchronous network and t_a parties in the asynchronous network and has the following properties.*

1. *Synchronous network:*

- (a) *t_s privacy: The view of the adversary is independent of the output triple shared on behalf of an honest dealer.*
- (b) *t_s correctness: Within time $T_{\text{VTS}} = T_{\text{VSS}} + T_{\text{Beaver}} + T_{\text{PrivRec}} + 3T_{\text{BC}} = T_{\text{VSS}} + 3T_{\text{BC}} + 2\Delta$, the honest parties output a degree- t_s Shamir-sharing of a multiplication triple on behalf of an honest dealer.*
- (c) *t_s strong commitment: If the dealer is corrupt, then either no honest party has an output, or all the honest parties output a degree- t_s Shamir-sharing of a multiplication triple on behalf of the dealer. Moreover, if some honest party computes its output at time T , then all the honest parties compute their output by time $T + 2\Delta$.*

2. *Asynchronous network:*

- (a) *t_a privacy: The view of the adversary is independent of the output triple shared on behalf of an honest dealer.*
- (b) *t_a correctness: Almost-surely, the honest parties eventually output a degree- t_s Shamir-sharing of a multiplication triple on behalf of an honest dealer.*
- (c) *t_a strong commitment: If the dealer is corrupt, then either no honest party has an output, or all the honest parties eventually output a degree- t_s Shamir-sharing of a multiplication triple on behalf of the dealer.*

Proof. We first prove the properties of Π_{VTS} in the synchronous network, followed by the proof for the asynchronous network.

1. *Synchronous network:*

- (a) *t_s privacy: In a synchronous network, for an honest dealer, each honest party computes its shares in Π_{VSS} by time T_{VSS} . All the parties thus begin the execution of Π_{Beaver} simultaneously, and by the guarantees of Π_{Beaver} , they receive the output within time Δ , that is each honest party computes its output of Π_{Beaver} by time $T_{\text{VSS}} + \Delta$. Further, by the guarantees of Π_{privRec} , we have that each honest party P_i receives its points $X(i), Y(i), Z(i)$ within time $T_{\text{VSS}} + 2\Delta$ and broadcasts $\text{OK}(i)$. Since the honest parties start their broadcast simultaneously, all honest parties (including the dealer) receive the $\text{OK}(i)$ messages by time $T_{\text{VSS}} + 2\Delta + T_{\text{BC}}$. Moreover, no honest party broadcasts $\text{NOK}(i)$ when the dealer is honest. Thus, the dealer*

constructs its set OK which includes all the honest parties, which also ensures that $|\text{OK}| \geq n - t_s$. This guarantees that the set $\text{NOK} \subset \mathcal{P} \setminus \text{OK}$ does not include any honest party. Hence, the publicly reconstructed points $X(i), Y(i), Z(i)$ for each $i \in \text{NOK}$ correspond to points held by the corrupt parties. This implies that an adversary knows t_s points on each polynomial $X(\cdot), Y(\cdot), Z(\cdot)$ which are of degree $t_s, t_s, 2t_s$ respectively, thus ensuring one degree of freedom. Hence, we have that for every candidate output triple $(X(\beta), Y(\beta), Z(\beta))$, we have a corresponding input triple (a_k, b_k, c_k) for some $k \in \{1, \dots, m\}$ unknown to the adversary that is consistent with the adversary's view.

- (b) t_s correctness: Let the dealer be honest. Note that all the honest parties obtain the output of Π_{VSS} instantiated by an honest dealer within time T_{VSS} . This further implies that Π_{Beaver} and Π_{PrivRec} succeed for all the honest parties by time $T_{\text{VSS}} + T_{\text{Beaver}} + T_{\text{PrivRec}} = T_{\text{VSS}} + 2\Delta$. Hence, each honest party P_i broadcasts $\text{OK}(i)$, which, by the validity of broadcast in the synchronous network, is received by all the honest parties, including the dealer by time $T_{\text{VSS}} + T_{\text{BC}} + 2\Delta$. Hence, all the honest parties simultaneously proceed to the next step at time $T_{\text{VSS}} + T_{\text{BC}} + 2\Delta$. Parties additionally keep broadcasting their shares corresponding to every $\text{NOK}(j)$ which is received. By the validity property of broadcast in the synchronous network, we also have that the dealer's broadcast of (OK, NOK) sets will be received by all the parties by time $T_{\text{VSS}} + 2T_{\text{BC}} + 2\Delta$. Finally, parties broadcast their shares corresponding to every $j \in \text{NOK}$. Again, by the validity and liveness of broadcast in the synchronous network, we have that every honest party's shares will be received by all the honest parties by time $T_{\text{VSS}} + 3T_{\text{BC}} + 2\Delta$. Further, we have that if $\text{NOK}(j)$ was broadcasted by some corrupt P_j and $X(j), Y(j), Z(j)$ is reconstructed by this time, then it would hold that $X(j) \cdot Y(j) = Z(j)$ and hence the dealer is not discarded. Thus, we have that all the honest parties output their shares by time $T_{\text{VSS}} + 3T_{\text{BC}} + 2\Delta$.
- (c) t_s strong commitment: If no honest party computes an output in the protocol then strong commitment holds trivially. Hence, we consider the case when there exists some honest party which computes an output. Note first that to ensure the correctness of the output, that is, to ensure that the honest parties output shares of a multiplication triple, it is required to verify that $X(\cdot) \cdot Y(\cdot) = Z(\cdot)$ holds for at least $2t_s + 1$ distinct points on these polynomials. In the protocol, this translates to ensuring that the relation holds for (at least) $2t_s + 1$ honest parties. Suppose there exists some honest P_h party that successfully outputs its shares in the protocol

without discarding the dealer. For contradiction, suppose that the relation does not hold for some honest party P_i . First, observe that since P_h outputs its shares, it implies that P_h computes the outputs of all the Π_{VSS} instances initiated by the dealer. Suppose the time at which P_h computed this is T . Note that every honest party would thus have computed its output by time $T + 2\Delta$ in the worst case. Suppose the worst case, that is P_i computed its output in Π_{VSS} at time $T + 2\Delta$. It is thus possible that P_i received its $X(i), Y(i), Z(i)$ at time $T + 2\Delta + T_{\text{Beaver}} + T_{\text{PrivRec}} = T + 4\Delta$, whereas P_h obtained its $X(h), Y(h), Z(h)$ at time $T + T_{\text{Beaver}} + T_{\text{PrivRec}} = T + 2\Delta$. That is, it is possible that some honest parties broadcast their $\text{OK}(i)$ or $\text{NOK}(i)$ after a 2Δ delay compared to other honest parties. Specifically, P_h may have broadcast $\text{OK}(h)$ and proceed to the next step by time $T + T_{\text{BC}} + 2\Delta$. Moreover, P_h 's broadcast would have been received by all within this time. In contrast, P_i 's broadcast may be delivered to parties (including P_h) by time $T + T_{\text{BC}} + 4\Delta$. This implies that $X(i), Y(i), Z(i)$ would have been reconstructed by time $T + 2T_{\text{BC}} + 4\Delta$. However, the earliest P_h can compute its output is at time $T + 3T_{\text{BC}} + 2\Delta$. This is because P_h waits for T_{BC} time for the dealer's broadcast of (OK, NOK) . It waits for another T_{BC} time for ensuring reconstruction of values corresponding to parties in NOK . Since we have that $T + 3T_{\text{BC}} + 2\Delta > T + 2T_{\text{BC}} + 4\Delta$, P_h must have received $X(i), Y(i), Z(i)$ before it proceeded to compute its output. If indeed $X(i) \cdot Y(i) = Z(i)$ did not hold, then P_h would have discarded the dealer, which is a contradiction. Thus, it must be that $X(i) \cdot Y(i) = Z(i)$. This also implies that every honest party P_i 's $\text{NOK}(i)$ would have been received by time at most $T + 2T_{\text{BC}} + 4\Delta$ and verified by P_h . Since P_h did not discard the dealer, $X(i) \cdot Y(i) = Z(i)$ must hold for every honest P_i . Given that the number of honest parties is at least $2t_s + 1$, $X(\cdot) \cdot Y(\cdot) = Z(\cdot)$ must hold. Consequently, no honest party will discard the dealer and hence all output their shares on the dealer's polynomials. Moreover, if some honest party computes its output by time T' , we have that it received all the corresponding broadcasts by time T' . By the fallback validity of broadcast, all the honest parties receive the necessary broadcasts by time $T' + 2\Delta$ and subsequently compute the output.

2. Asynchronous network:

- (a) t_a privacy: In the asynchronous network, for an honest dealer, each honest party computes its shares in Π_{VSS} eventually. This ensures that all the parties eventually begin the execution of Π_{Beaver} and receive their output. Further, this also guarantees that parties invoke Π_{privRec} , and each honest party P_i receives its points

$X(i), Y(i), Z(i)$ eventually and broadcasts $\text{OK}(i)$. Thus we have that even if the corrupt parties are silent, an honest dealer can compute the set OK of size (at least) $n - t_s$ eventually. In the worst case, the set NOK broadcasted by the dealer may be of size (at most) $(t_s - t_a)$ and moreover may comprise completely of honest parties. Note that the points $X(i), Y(i), Z(i)$ of each $i \in \text{NOK}$ are revealed publicly. This causes the adversary to learn $t_s - t_a$ points on each of $X(\cdot), Y(\cdot), Z(\cdot)$ corresponding to the $t_s - t_a$ honest parties included in NOK , in addition to the t_a points of the corrupt parties. The adversary thus learns t_s points on each of $X(\cdot), Y(\cdot), Z(\cdot)$, which is exactly the information available to the adversary in the synchronous setting. By the same argument as privacy in the synchronous setting, we have that the adversary's view is independent of the output multiplication triple.

- (b) t_a correctness: Let the dealer be honest. By the t_a correctness of Π_{VSS} in the asynchronous network, we have that the honest parties will eventually compute their output. Similarly, by the t_a correctness of Π_{Beaver} and Π_{privRec} , we also have that each honest P_i eventually receives its $X(i), Y(i), Z(i)$ and consequently broadcasts $\text{OK}(i)$. Since we have $n - t_a$ honest parties, it must hold that the dealer will indeed be able to construct the set OK consisting of at least $n - t_s$ parties. Again, due to the validity property of broadcast in the asynchronous network, it is ensured that the parties will receive the dealer's broadcast of (OK, NOK) , and consequently reconstruct the values $X(j), Y(j), Z(j)$. These reconstructed values are guaranteed to be correct. Moreover, every $\text{NOK}(j)$ broadcasted by a corrupt P_j will eventually be received by all the honest parties due to the consistency property of broadcast. Hence, we have that parties will eventually reconstruct $X(j), Y(j), Z(j)$ for each P_j and verify its correctness. Thus, all the honest parties will eventually output shares on the polynomials shared by the dealer as the shares of its multiplication triple.
- (c) t_a strong commitment: This follows similarly to the synchronous case due to the t_a strong commitment property of Π_{VSS} and consistency of Π_{BC} in the asynchronous network. Specifically, if no honest party computes its output, then commitment holds trivially. On the other hand, if any honest party computes its output in Π_{VSS} , then by the t_s strong commitment property, we have that all the honest parties compute their output. Similarly, by the t_a correctness property of Π_{privRec} and Π_{Beaver} , it is ensured that parties eventually compute their point on $X(\cdot), Y(\cdot), Z(\cdot)$. Now observe that every honest party computes its output only upon verifying that the multiplicative relation holds true for at least $n - t_a$ parties. Since all the com-

munication occurs via broadcast for the verification, the consistency property of broadcast ensures that if some honest party computes its output then eventually all the honest parties compute their output. Further, given that the adversary can corrupt at most t_a parties in the asynchronous network, this ensures that the multiplicative relation of $X(\cdot), Y(\cdot), Z(\cdot)$ is verified for at least $n - 2t_a \geq 2t_s + 1$ honest parties. As mentioned earlier, since $X(\cdot), Y(\cdot), Z(\cdot)$ are degree $t_s, t_s, 2t_s$ polynomials respectively, this verification ensures the correctness of the multiplication triples shared by a corrupt dealer.

□

6.9 Preprocessing Phase

6.9.1 Private Reconstruction Protocol

We now describe the reconstruction of a degree- t_s shared value $\langle v \rangle$ to a particular party P^* . For this, all the parties reveal their shares of $\langle v \rangle$ to P^* , who tries to recover the secret as follows. P^* waits for Δ time to receive the shares from other parties. P^* waits for $2t_s + 1$ shares, all of which lie on a degree- t_s polynomial. If such a polynomial is reconstructed, it is guaranteed to be correct since it agrees with the shares of at least $t_s + 1$ honest parties. Recovering such a polynomial requires P^* to apply error correction repeatedly in an “online” manner to recover the secret in the case of an asynchronous network. Whereas, in the synchronous network case, it is guaranteed that all the honest parties will send their shares lying on the same polynomial within Δ time, and hence, the reconstruction will succeed. Reconstruction towards all can be performed similarly with n instances of the protocol, one towards each party. Alternatively, parties can also broadcast their respective shares to reconstruct a value publicly.

Protocol 6.9.1: Π_{privRec}

Input: Parties hold the degree- t_s Shamir-sharing of a value $\langle v \rangle$.

Common Input: Description of a field \mathbb{F} , n non-zero distinct field elements $1, \dots, n$ and the identity of a party P^* .

1. Each P_i sends its share $\langle v \rangle_i$ to P^* .
2. P^* **waits for Δ time** and then applies online error correction on the received shares as follows. For each $r = 0, \dots, t_s$:
 - (a) Upon receiving $n - t_s \geq 2t_s + 1$ values, P^* looks for a codeword of a polynomial of degree- t_s with a distance of at most r from the values it received. If there is no

such codeword, then P^* proceeds to the next iteration. Otherwise, P^* sets $p_r(x)$ as the unique Reed-Solomon reconstruction.

- (b) If $p_r(j) = \langle v \rangle_j$ holds for at least $2t_s + 1$ parties, P^* computes $v = p_r(0)$. Otherwise, it proceeds to the next iteration.
-

Theorem 6.9.2. *Protocol Π_{privRec} is secure against an adversary corrupting up to t_s parties in the synchronous network and t_a parties in the asynchronous network and has the following properties.*

1. *Synchronous network:*

- (a) *t_s correctness: Within time Δ , each honest party outputs v .*

2. *Asynchronous network:*

- (a) *t_a correctness: Each honest party eventually outputs v .*

Proof. We prove the properties of Π_{privRec} in the synchronous network and subsequently in the asynchronous network.

1. *Synchronous network:*

- (a) *t_s correctness:* In a synchronous network, each honest party receives shares on the degree- t_s polynomial from every other honest party within time Δ . Thus, an honest party receives at least $n - t_s \geq 2t_s + 1$ correct points on the polynomial. Moreover, if an honest party receives r incorrect shares by time Δ , then by the guarantees of Reed-Solomon codes and given that $r \leq t_s$, an honest party having (at least) $2t_s + 1 + r$ points can correct r points and recover the correct polynomial.

2. *Asynchronous network:*

- (a) *t_a correctness:* In the asynchronous network, note that each honest party will eventually receive $n - t_a \geq 2t_s + t_a + 1$ correct points from the honest parties. By reasoning similar to that in the synchronous setting, the honest parties will eventually compute the correct polynomial defined by the honest parties' shares.

□

6.9.2 Beaver's Multiplication Protocol

This protocol uses the well-known Beaver's circuit randomization [23] technique to perform the multiplication of two shared values. Specifically, given a pre-shared random and private

multiplication triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, this technique reduces the computation of $\langle z \rangle = \langle xy \rangle$ from $\langle x \rangle$ and $\langle y \rangle$ to two public reconstructions. Towards this, parties first locally compute $\langle d \rangle = \langle x \rangle - \langle a \rangle$ and $\langle e \rangle = \langle y \rangle - \langle b \rangle$, followed by public reconstruction of d and e . Now, parties can compute $\langle z \rangle$ locally using these values together with the shared multiplication triple. More precisely, since $z = xy = ((x - a) + a)((y - b) + b) = (d + a)(e + b) = de + db + ea + ab = de + db + ea + c$ parties can compute $\langle z \rangle = \langle xy \rangle = de + d\langle b \rangle + e\langle a \rangle + \langle c \rangle$. The formal description of the protocol appears below.

Protocol 6.9.3: Π_{Beaver}

Input: Parties hold the degree- t_s Shamir-sharing of a triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ and the inputs $\langle x \rangle$ and $\langle y \rangle$.

1. Parties locally compute $\langle d \rangle = \langle x \rangle - \langle a \rangle$ and $\langle e \rangle = \langle y \rangle - \langle b \rangle$.
 2. Parties execute $2n$ instances of Π_{privRec} , two towards every party for reconstructing d and e respectively. **Wait for time Δ .**
 3. Parties locally compute $\langle z \rangle = de + d\langle b \rangle + e\langle a \rangle + \langle c \rangle$.
-

Theorem 6.9.4. *Protocol Π_{Beaver} is secure against an adversary corrupting up to t_s parties in the synchronous network and t_a parties in the asynchronous network and has the following properties.*

1. *Synchronous network:*
 - (a) *Liveness:* At time Δ , every honest party has an output.
 - (b) *t_s privacy:* If (a, b, c) is a random multiplication triple from the adversary's view, then the view of the adversary is independent of x and y (and thus z).
 - (c) *t_s correctness:* Within time Δ , the honest parties output a degree- t_s Shamir-sharing of z such that $z = xy$ if and only if (a, b, c) is a correct multiplication triple, i.e. $c = ab$ holds.
2. *Asynchronous network:*
 - (a) *Liveness:* Every honest party eventually has an output.
 - (b) *t_s privacy:* If (a, b, c) is a random multiplication triple from the adversary's view, then the view of the adversary is independent of x and y (and thus z).
 - (c) *t_a correctness:* The honest parties eventually output a degree- t_s Shamir-sharing of z such that $z = xy$ if and only if (a, b, c) is a correct multiplication triple, i.e. $c = ab$ holds.

Proof. We prove the properties of Π_{Beaver} in both networks simultaneously.

1. Liveness: By the linearity property of Shamir-sharing, parties can locally compute the degree- t_s Shamir-sharing of d and e . Further, due to t_s (resp. t_a) correctness of Π_{privRec} in the synchronous (resp. asynchronous) network, we have that parties will receive the reconstructed values d and e within time Δ (resp. eventually). The computation of $\langle z \rangle$ is local thereafter; hence, we have the required liveness guarantees.
2. t_s (resp. t_a) privacy: If (a, b, c) is a random multiplication triple from the adversary's view, then for every possible x and y values, there exist a and b such that they are consistent with the adversary's view and the publicly reconstructed values of d and e . Thus, the adversary's view is independent of x and y (hence z).
3. t_s (resp. t_a) correctness: Note that $z = de + db + ea + c = (x - a)(y - b) + (x - a)b + (y - b)a + c = xy + c - ab$. Hence, by inspection, it is clear that $z = xy$ if and only if $c - ab = 0$, that is, $c = ab$.

□

6.9.3 Triple Extraction Protocol

The last component of the Beaver triple generation phase of our protocol is a triple extraction protocol that consumes one (verified) multiplication triple, say $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$, shared by each party $P_i \in \text{Com}$ in the prior stage and extracts $h + 1 - t$ random triples not known to any party, where $h = \lfloor \frac{|\text{Com}| - 1}{2} \rfloor$. For simplicity, let $m = |\text{Com}|$ and without loss of generality, we assume $\text{Com} = \{P_1, \dots, P_m\}$. At a high level, the protocol proceeds as follows. First, the parties “transform” the m random shared triples $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$ for each $i \in \{1, \dots, m\}$ into m correlated triples $(\langle x_i \rangle, \langle y_i \rangle, \langle z_i \rangle)$ for every $i \in \{1, \dots, m\}$ such that the values $\{x_i, y_i, z_i\}_{i \in \{1, \dots, m\}}$ lie on the polynomials $X(\cdot)$, $Y(\cdot)$ and $Z(\cdot)$ of degree h , h and $2h$ respectively where $X(\cdot) \cdot Y(\cdot) = Z(\cdot)$. Specifically, for each $i \in \{1, \dots, m\}$, it holds that $X(i) = x_i$, $Y(i) = y_i$ and $Z(i) = z_i$ where $1, \dots, m$ are publicly known distinct elements from \mathbb{F} . Furthermore, the transformation ensures that the adversary knows $\{x_i, y_i, z_i\}$ only if P_i is corrupt. This implies that the adversary may know (at most) t points on each of the polynomials $X(\cdot)$, $Y(\cdot)$ and $Z(\cdot)$ of degree h , h and $2h$ respectively, thus guaranteeing a degree of freedom of $h + 1 - t$ in $X(\cdot)$, $Y(\cdot)$ (and thus $Z(\cdot)$). Parties thus output the shared evaluation of these polynomials at $h + 1 - t$ publicly known points $\beta_1, \dots, \beta_{h+1-t}$ as the extracted shared multiplication triples.

The transformation itself works as follows. The parties simply set $x_i = a_i$, $y_i = b_i$, $z_i = c_i$ for $i \in \{1, \dots, h + 1\}$. Next, $\langle x_i \rangle$ and $\langle y_i \rangle$ for every $i \in \{h + 2, \dots, m\}$ can be computed *non-interactively* by taking linear combination of $\{x_i, y_i\}_{i \in \{1, \dots, h+1\}}$. Following this, $\langle z_i \rangle$ for

every $i \in \{h + 2, \dots, m\}$ is computed using Beaver's trick where the inputs are $\langle x_i \rangle$ and $\langle y_i \rangle$ and the random multiplication triple consumed is $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$. Clearly, if P_i is corrupt, then x_i, y_i, z_i is known to the adversary as claimed. Finally, we note that triple extraction reduces to running a batch of $\mathcal{O}(h)$ Beaver multiplications. The formal description appears in Protocol 6.9.5.

Protocol 6.9.5: Triple Extraction – $\Pi_{\text{tripleExt}}$

Common input: The description of a field \mathbb{F} , a set $\text{Com} \subseteq \mathcal{P}$ such that $m = |\text{Com}| \geq n - t_s$, $m = 2h + 1$ non-zero distinct elements $1, \dots, m$ and $h + 1 - t_s$ non-zero distinct elements $\beta_1, \dots, \beta_{h+1-t_s}$. Without loss of generality, assume $\text{Com} = \{P_1, \dots, P_m\}$.

Input: Parties hold the degree- t_s shared triples $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$ for every $i \in \{1, \dots, m\}$ such that (a_i, b_i, c_i) is known to party P_i .

1. For each $i \in \{1, \dots, h + 1\}$, parties locally set $\langle x_i \rangle = \langle a_i \rangle$, $\langle y_i \rangle = \langle b_i \rangle$ and $\langle z_i \rangle = \langle c_i \rangle$.
 2. Let $X(\cdot)$ and $Y(\cdot)$ be the degree- h polynomials defined by the points $\{x_i\}_{i \in \{1, \dots, h+1\}}$ and $\{y_i\}_{i \in \{1, \dots, h+1\}}$ respectively such that $X(i) = x_i$ and $Y(i) = y_i$ for all $i \in \{1, \dots, h + 1\}$.
 3. For each $i \in \{h + 2, \dots, m\}$, parties locally compute $\langle x_i \rangle = \langle X(i) \rangle$ and $\langle y_i \rangle = \langle Y(i) \rangle$.
 4. Parties invoke Π_{Beaver} with $\{\langle x_i \rangle, \langle y_i \rangle, \langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle\}_{i \in \{h+2, \dots, m\}}$ and obtain $\{\langle z_i \rangle\}_{i \in \{h+2, \dots, m\}}$ where $z_i = x_i y_i$ for every $i \in \{h + 2, \dots, m\}$. **Wait for time Δ .**
 5. Let $Z(\cdot)$ be the degree- $2h$ polynomial defined by the points $\{z_i\}_{i \in \{1, \dots, m\}}$ such that $Z(i) = z_i$ for all $i \in \{1, \dots, m\}$.
 6. Parties locally compute $\langle \mathbf{a}_i \rangle = \langle X(\beta_i) \rangle$, $\langle \mathbf{b}_i \rangle = \langle Y(\beta_i) \rangle$ and $\langle \mathbf{c}_i \rangle = \langle Z(\beta_i) \rangle$ for every $i \in \{1, \dots, h + 1 - t_s\}$.
-

Theorem 6.9.6. *Protocol $\Pi_{\text{tripleExt}}$ is secure against an adversary corrupting up to t_s parties in the synchronous network and up to t_a parties in the asynchronous network and has the following properties.*

1. *Synchronous network:*
 - (a) t_s privacy: *The triples $\{(\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i)\}_{i \in \{1, \dots, h+1-t_s\}}$ are random from the adversary's view.*
 - (b) t_s correctness: *Within time $T_{\text{tripleExt}} = \Delta$, the honest parties output a degree- t_s Shamir-sharing of each triple $\{(\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i)\}_{i \in \{1, \dots, h+1-t_s\}}$.*
2. *Asynchronous network:*

- (a) t_s privacy: The triples $\{(\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i)\}_{i \in \{1, \dots, h+1-t_s\}}$ are random from the adversary's view.
- (b) t_a correctness: The honest parties eventually output a degree- t_s Shamir-sharing of each triple $\{(\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i)\}_{i \in \{1, \dots, h+1-t_s\}}$.

Proof. We prove the properties of $\Pi_{\text{tripleExt}}$ in both networks simultaneously.

1. t_s privacy: Note that (since $t_a < t_s$) in the worst case, there will be at most t_s corrupt parties in the set Com. This implies that at most t_s points are known to the adversary on each of the polynomials $X(\cdot)$ and $Y(\cdot)$. This ensures a degree of freedom of $h+1-t_s$ on each of these polynomials (and hence on $Z(\cdot)$). Hence, we have that for every candidate set of triples $\{(\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i)\}_{i \in \{1, \dots, h+1-t_s\}}$, there exists a set of $h+1-t_s$ corresponding candidate input triples $(\langle a_j \rangle, \langle b_j \rangle, \langle c_j \rangle)$ unknown to the adversary that is consistent with the adversary's view.
2. t_s (resp. t_a) correctness: By the properties of Π_{Beaver} in the synchronous (resp. asynchronous) network, we have that parties will receive obtain $\{\langle z_i \rangle\}_{i \in \{h+2, \dots, m\}}$ within time Δ (resp. eventually). Since all the input triples are guaranteed to be valid multiplication triples, by construction of the protocol, it holds that $Z(\cdot) = X(\cdot) \cdot Y(\cdot)$ such that $X(\cdot), Y(\cdot)$ are degree- h polynomials and $Z(\cdot)$ is a degree- $2h$ polynomial. It thus follows that all the honest parties output $(\langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle) = (\langle X(\beta_i) \rangle, \langle Y(\beta_i) \rangle, \langle Z(\beta_i) \rangle)$ for every $i \in \{1, \dots, h+1-t_s\}$ within time Δ (resp. eventually). Moreover, the relation $\mathbf{c}_i = \mathbf{a}_i \cdot \mathbf{b}_i$ holds for every $i \in \{1, \dots, h+1-t_s\}$ since $Z(\cdot) = X(\cdot) \cdot Y(\cdot)$ holds.

□

6.10 The Complete MPC Protocol

This section describes our complete MPC protocol as a composition of the primitives described so far and the existing primitives detailed in Section 6.4. It has the following well-known two-phase structure: a preprocessing phase wherein parties generate random Beaver triples and an online phase wherein parties consume these triples to evaluate the circuit. We elaborate on these two phases below.

Beaver Triple Generation. In this phase, the goal is to generate degree- t_s shares of random multiplication triples of the form (a, b, c) where $c = a \cdot b$. We require C random triples to be shared to evaluate a circuit with C multiplication gates. This phase can be further viewed as consisting of three stages:

1. Triples with a dealer: In this stage, each party P_i acts as a dealer and shares triples of the form (a_i, b_i, c_i) such that $c_i = a_i \cdot b_i$ must hold. The dealer is required to provide a perfect

zero-knowledge proof to establish the correctness of its triples. Our main contribution lies in this stage, where the sharing of triples is performed using the verifiable secret sharing protocol (Π_{VSS}) discussed in Section 6.7. Further, we also give a protocol for verifiable triple sharing (Π_{VTS}), which allows the dealer to prove that the triples it shared are indeed correct. If a dealer’s sharing fails, then its triples are ignored by all the parties. This protocol appears in Section 6.8.

2. Agreement on a Common Set (ACS): Irrespective of the network type, we have that the triple sharing instances of the honest dealers will eventually terminate. However, the instances corresponding to t_s corrupt dealers in the synchronous network, and analogously t_a corrupt dealers in the asynchronous network may never terminate. Further, parties are unaware of the underlying network condition, and in the worst case, t_s corrupt parties may not even initiate their triple sharing. To prevent endless waiting, parties proceed upon successful completion of (at least) $n - t_s$ instances of triple sharing. Since parties may receive messages in different order, we need to ensure that all the parties agree on the set of parties for whom triple sharing is successfully completed. This task is handled by the ACS protocol, Π_{ACS} , described in Section 6.4.
3. Triples without a dealer: Once a common set of parties Com whose triple sharing has terminated successfully been determined, the goal is to then extract random triples unknown to any party. For this, we use the existing triple sharing protocol, $\Pi_{\text{tripleExt}}$, which consumes the triples shared by each party in Com and extracts random triples.

Circuit Evaluation. This is the second phase of our MPC protocol, which at a high level, consists of four stages. At the input sharing stage, parties share their inputs to the circuit. Following this, parties run Π_{ACS} to agree on a common set of at least $n - t_s$ parties that have provided input to the MPC protocol. The second stage comprises of the shared evaluation of the circuit. Since our sharing is linear, addition and multiplication by a constant operations can be performed locally. For multiplication, we rely on the well-known technique of Beaver’s circuit randomisation [23]. Here, parties use the triples generated in the prior phase to evaluate multiplication gates in the circuit using Beaver multiplication. In this protocol, by using a pre-shared triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, the task of computing a degree- t_s sharing $\langle xy \rangle$ from $\langle x \rangle$ and $\langle y \rangle$ reduces to two public reconstructions. The protocol description for Beaver’s multiplication protocol, Π_{Beaver} , appears in Section 6.4. The third stage corresponds to the reconstructing the output of the circuit to the parties. Finally, the last stage ensures that sufficiently many parties have obtained the same output. If this holds, then parties safely terminate with the output, in the MPC protocol as well as all the underlying protocols. This

concludes our MPC protocol, which appears below. In the protocol description, we perform input sharing along with triple sharing in the first phase and invoke Π_{ACS} to decide on a common set of parties that successfully share both. Hence, we avoid an additional invocation of Π_{ACS} during the circuit evaluation phase.

Protocol 6.10.1: Network-Agnostic MPC – $\Pi_{\text{MPC}}^{\text{na}}$

Common input: The description of a circuit, the field \mathbb{F} , n non-zero distinct elements $1, \dots, n$ and a parameter h where $n - t_s = 2h + 1$. Let $m = \lceil \frac{C}{h+1-t_s} \rceil$.

Input: Parties hold their inputs (belonging to $\mathbb{F} \cup \{\perp\}$) to the circuit.

(Beaver triple generation and Input sharing:)

1. **(Beaver Triple generation with a dealer)** Each P_i chooses m random multiplication triples and executes m instances of Π_{VTS} (Protocol 6.8.1, Section 6.8) simultaneously.
2. **(Input sharing)** Each party P_i holding k_i inputs to the circuit executes k_i instances of Π_{VSS} simultaneously (Protocol 6.7.1, Section 6.7). Parties **wait for time** T_{VTS} .
3. **(Input to ACS)** Each P_i initialises a set $S_i \leftarrow \phi$. It includes j in S_i if it receives an output in all the Π_{VSS} and Π_{VTS} instances of P_j .
4. **(ACS Execution)** Parties invoke Π_{ACS} (Protocol 6.4.9, Section 6.4) to agree on a set Com of at least $n - t_s$ parties whose instances of triple sharing and input sharing will terminate eventually for all the honest parties. Let $(\langle a_i^j \rangle, \langle b_i^j \rangle, \langle c_i^j \rangle)$ for $j \in [m]$ denote the triples shared by $P_i \in \text{Com}$. The input sharing for the parties outside Com is taken as default sharing of 0. Parties **wait for time** T_{ACS} .
5. **(Beaver Triple Extraction)** Upon receiving output from Π_{ACS} , parties execute m instances of $\Pi_{\text{tripleExt}}$ (Protocol 6.9.5, Section 6.4) with Com as the common input and additionally $(\langle a_i^j \rangle, \langle b_i^j \rangle, \langle c_i^j \rangle)$ for every $P_i \in \text{Com}$ as the input for the j^{th} instance. Let $(\langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle)$ for $i \in [C]$ denote the random multiplication triples generated. **Wait for time** Δ .

(Circuit evaluation:)

1. **(Linear Gates)** Parties locally apply the linear operation on their respective shares of the inputs.
2. **(Multiplication Gates)** Let $(\langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle)$ be the multiplication triple associated with the i^{th} multiplication gate with shared inputs $(\langle x_i \rangle, \langle y_i \rangle)$. Parties invoke Π_{Beaver} (Protocol 6.9.3, Section 6.4) with $\{\langle x_i \rangle, \langle y_i \rangle, \langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle\}$ for all gates i at the same layer of the circuit and obtain the corresponding $\langle z_i \rangle$ as the output sharing for every gate i . **Wait for time** Δ .

3. **(Output)** For an output gate y with the associated sharing $\langle y \rangle$, upon computing the share of y , parties execute Π_{privRec} (Protocol 6.9.1, Section 6.4) towards every party P_i .
Wait for time Δ .
 4. **(Termination:)** Each party P_i does the following:
 - If y has been computed during the output step, then send (ready, y) to all the parties.
 - If (ready, y) has been received from at least $t_s + 1$ distinct parties, then send (ready, y) to all the parties, if not sent before.
 - If (ready, y) has been received from at least $2t_s + 1$ distinct parties, then output y and terminate the protocol.
-

Theorem 6.10.2. *Let n, t_s, t_a be such that $t_a < t_s$ and $n > 2t_s + \max(2t_a, t_s)$. Protocol 6.10.1, Π_{MPC} , is a network-agnostic MPC protocol that is perfectly-secure against an adversary corrupting up to t_s parties in a synchronous network and up to t_a parties in the asynchronous network. It has the following properties:*

- t_s correctness: *In a synchronous network, all the honest parties compute $y = f(x_1, \dots, x_n)$, where $x_i = 0$ if $i \notin \text{Com}$ such that $|\text{Com}| \geq n - t_s$ and every honest party belongs to Com within time $T_{\text{MPC}} = T_{\text{VTS}} + T_{\text{ACS}} + T_{\text{tripleExt}} + D \cdot T_{\text{Beaver}} + T_{\text{PrivRec}}$.*
- t_a correctness: *When the network is asynchronous, all the honest parties eventually compute $y = f(x_1, \dots, x_n)$, where $x_i = 0$ if $i \notin \text{Com}$ such that $|\text{Com}| \geq n - t_s$.*
- t_s privacy: *Irrespective of the network type, the adversary's view is independent of the inputs of the honest parties in Com .*

Proof. We first consider a synchronous network with up to t_s corruptions. By the t_s correctness property of the triple sharing and verifiable secret sharing protocols, we have that the triple sharing and input sharing instances of all the honest parties will terminate within time T_{VTS} . Further, this implies that the input requirements of the protocol Π_{ACS} for synchronous network will hold true. Hence, by the t_s correctness property of Π_{ACS} , within time T_{ACS} parties will output a set Com such that $|\text{Com}| \geq n - t_s$ and it includes all the honest parties. Moreover, if there is some corrupt $P_i \in \text{Com}$, it implies that some honest party P_h computed the output of verifiable secret sharing and triple sharing in P_i 's instances. If not, then it would mean that no honest party includes P_i in its set S_i , and hence, all the parties would input 0 for the instance Π_{BA}^i . By the validity property of Π_{BA} in the synchronous network, we have that parties

output 0 in the instance Π_{BA}^i . Thus, P_i is excluded from Com , which is a contradiction. Therefore, given that some honest party computes the output in P_i 's instances of verifiable secret sharing and triple sharing, by the t_s strong commitment property of both these protocols, we have that all the honest parties compute an output. Consequently, we have that parties hold shares corresponding to m multiplication triples shared by each party in Com . Subsequently, by the t_s correctness property of $\Pi_{\text{tripleExt}}$, within time Δ parties will compute the shares of random triples for $h + 1 - t_s$ for each instance of $\Pi_{\text{tripleExt}}$. Given that we have $m = \lceil \frac{C}{h+1-t_s} \rceil$, parties obtain the random shares for C multiplication triples. In the circuit evaluation phase, the linear gates are computed locally. Whereas for the multiplication gates, the t_s correctness property of Π_{Beaver} ensures that all the honest parties obtain the correct sharing of the output of the gates within time Δ . Finally, the t_s correctness of the reconstruction protocol Π_{privRec} in the synchronous network ensures that parties receive their output within time Δ . Thus, we have that in a synchronous network, all the honest parties will send (ready, y) messages. Since there are at least $2t_s + 1$ honest parties, termination is guaranteed. The proof for the t_s correctness in the asynchronous network follows similarly, with the modification that it now relies on the t_s correctness of all the subprotocols in the asynchronous network. For termination, note that at least $2t_s + 1$ honest parties will eventually send (ready, y) message which all the honest parties will receive. Moreover, if some honest party terminates with an output y , then it implies that it received (ready, y) from at least $t_s + 1$ honest parties. All honest parties will eventually receive these messages and send (ready, y) to all. Since there are at least $2t_s + 1$ honest parties, termination is ensured.

The t_s privacy of the MPC protocol in either of the network conditions follows from the t_s privacy of the subprotocols. Specifically, from the t_s privacy of Π_{VSS} , we have that the inputs of honest parties are random from the adversary's view. Further, from the t_s privacy of Π_{VTS} , it follows that the multiplication triples shared by each honest P_i for $i \in \text{Com}$ are random from the adversary's view. Given this, the t_s privacy of $\Pi_{\text{tripleExt}}$ ensures that the multiplication triples extracted from the triples of parties in Com are indeed random from the view of the adversary. Finally, the t_s privacy of Π_{Beaver} guarantees that the adversary does not learn any additional information during the evaluation of a multiplication gate. Moreover, the rest of the gates are computed non-interactively, thus ensuring t_s privacy of the MPC protocol. \square

Chapter 7

Conclusion and Open Problems

This thesis settles some of the long-standing open questions in the domain of perfectly-secure multiparty computation. Specifically, the work in this thesis extends over various network settings which include the synchronous, asynchronous and network-agnostic. In the former two categories this thesis offers communication efficient protocols, whereas it establishes impossibility and feasibility results in the latter category.

In the synchronous network model, we first considered the primitive of broadcast owing to the fact that its implementation is one of the primary communication bottlenecks in MPC protocols. Here, we identified that typically, broadcast protocol implementations are of two flavours: the first category of protocols has a round complexity of $\theta(n)$, whereas the second category leverages randomization to achieve expected constant round protocols. This thesis focused on the latter with the end goal of MPC protocols where the round complexity (in expectation) is independent of the number of parties n . Although protocols existed in this regime, those constructions had large communication complexity, specifically $\mathcal{O}(n^2L + n^6 \log n)$ expected number of bits transmitted for broadcasting a message of length L . This led to a significant communication blowup in secure computation protocols in this setting. In this thesis, we substantially improved the communication complexity of broadcast in constant expected time to $\mathcal{O}(nL + n^4 \log n)$ for broadcasting L bits. We also constructed a parallel broadcast protocol, where n parties wish to broadcast L bit messages in parallel. Our protocol has no asymptotic overhead for $L = \Omega(n^2 \log n)$, which is a common communication pattern in perfectly secure MPC protocols, thereby paving a way for efficient secure computation.

Subsequently, this thesis contributed to communication efficiency of generic secure computation. Analogous to broadcast, we identified two classes of MPC protocols: the first class comprises of protocols which have a round complexity of $\Omega(D + n)$, whereas the second class

has protocols with $\mathcal{O}(D)$ in expectation for depth D circuits. While a communication complexity of $\mathcal{O}(n \log n)$ per gate was achieved for the former category nearly 15 years ago, the second category lagged significantly behind at $\mathcal{O}(n^4 \log n)$ cost per gate. The work in this thesis bridged the gap between these two classes of protocols and proved that it is simultaneously possible to achieve the best of both the classes. Compared to state-of-the-art protocols in the former class, for a circuit with $C > n^3$ gates and depth $D \ll n$, our results significantly improved the run time from $\Omega(n + D)$ to expected $\mathcal{O}(D)$ while keeping communication complexity at $\mathcal{O}(Cn \log n)$. Whereas compared to state-of-the-art protocols in the latter class, for $C > n^3$, our results significantly improve the communication complexity from $\mathcal{O}(Cn^4 \log n)$ to $\mathcal{O}(Cn \log n)$ while keeping the expected run time at $\mathcal{O}(D)$.

Finally, we targeted the design of communication efficient protocols from the perspective of the more realistic model of an asynchronous network. In contrast to the synchronous network where $\mathcal{O}(Cn \log n)$ cost protocols were known for nearly 15 years, albeit the $\Omega(n + D)$ round complexity, the landscape of asynchronous protocols was vastly different. Despite the 30 years of research, protocols in the asynchronous setting required $\Omega(Cn^2 \log n)$ communication complexity. Our work in this thesis closed this gap between synchronous and asynchronous secure computation and showed the first asynchronous protocol with $\mathcal{O}(Cn \log n)$ communication complexity.

In our concluding work, deviating from the focus on a monolithic view of the network and communication efficiency of protocols, we established the feasibility of network-agnostic perfectly-secure MPC. We identified that the network-agnostic setting is apt for scenarios where parties are unaware of the network type, and yet wish to achieve the best security guarantees that are possible. While the lower bound on the threshold for synchronous and asynchronous networks was already known for more than three decades, the same question remained open for the network-agnostic setting. The work in this thesis proved that the bound of $n > 3t_s + t_a$ conjectured by the prior works is not tight. We established a new lower bound of $n > 2 \max(t_s, t_a) + \max(t_s, 2t_a)$ for perfectly-secure network-agnostic MPC and proved its sufficiency by constructing a protocol matching this optimal resilience.

In conclusion, this thesis changed the landscape of perfectly-secure multiparty computation. On the one hand, it has advanced the research on the communication complexity of protocols in the well-studied, synchronous and asynchronous network models. On the other, it has advanced the nascent research area of network-agnostic protocols by establishing feasibility and impossibility results.

7.1 Open Problems

This thesis also spawns some interesting open questions, some of which have been targeted by recent follow-up works. We first note the open questions in the synchronous network model, followed by the asynchronous network. We conclude with potential research directions in the network-agnostic setting.

Synchronous Network. In the domain of expected constant round broadcast protocols, our work achieves $\mathcal{O}(nL + n^4 \log n)$ communication complexity for broadcast an L -bit message. Whereas our parallel broadcast costs $\mathcal{O}(n^2L + n^4 \log n)$. Although this was asymptotically optimal from the perspective of our multiparty computation protocol where each party required to broadcast $\mathcal{O}(n^2 \log n)$ bits, it was a good open problem to consider improving the L -independent term of our communication complexity. A recent work of [17] targets this problem, and we encourage the readers to refer to this paper for further details. Another interesting problem in the domain of synchronous network model is to identify the trade-off between threshold of corruption and communication complexity. Specifically, our work achieves $\mathcal{O}(n \log n)$ communication per multiplication gate in the setting of $t < n/3$, which is optimal up to logarithmic factors. However, the known lower bound for the case of $t < n/4$ in the synchronous setting hints at protocols with $\mathcal{O}(\log n)$ protocols, where the logarithmic factor is due to the size of field [58]. That is, while our protocol in the $t < n/3$ setting has a linear overhead per gate, we believe that with a lower threshold of $t < n/4$, it should be possible to achieve constant overhead. This is an interesting open direction to pursue and either obtain a protocol or improve the existing lower bound. Another interesting line of research is to solve analogous questions for statistical security. Similar to the case of perfectly-secure protocols, linear communication per gate has been obtained with $\Omega(n + D)$ rounds [77]. However, the question of obtaining linear communication protocols with $\mathcal{O}(D)$ rounds in the statistical setting has been open for a long time. In contrast to perfect security, the optimal threshold for generic secure computation in the statistical setting is known to be $t < n/2$. Additionally, in this setting, a broadcast channel assumption is known to be necessary which has to be accounted for while designing protocols.

Asynchronous Network. In the asynchronous network model with optimal resilience of $t < n/4$ for perfect security, this thesis provides a secure computation protocol with $\mathcal{O}(Cn \log n)$ communication complexity. Although a communication of linear (in the number of parties n) number of field elements seems like a natural barrier, a lower bound for this remains unknown. One interesting question would be to resolve this by either establishing a lower bound on the communication complexity which matches the cost of our protocol,

or by designing a protocol with improved communication complexity. Similar to the case of the synchronous network, another natural question here is that of achieving linear communication complexity protocols for statistical security, where the optimal corruption threshold is $t < n/3$. Although this problem has been recently solved in [78, 81], the communication complexity of their protocol has a very high circuit size independent factor of n^{14} . Thus, tackling the same problem with an improved complexity stays open.

Network-agnostic. Our work in the network-agnostic setting targets the feasibility and impossibility of secure computation, and hence does not aim for communication and computational efficiency. Consequently, our upper bound result, which is the network-agnostic protocol, requires exponential computation due to the necessity of a clique-finding algorithm. One interesting potential question is to replace the clique finding with some techniques which require polynomial time, or redesign the upper bound protocol completely to achieve a polynomial time solution. Being a relatively new area of research, most of the works have focused on establishing feasibility of network-agnostic protocols. Owing to this, communication efficiency has not been a parameter at the forefront during the design of these protocols. Focusing on communication efficient protocols in this setting opens avenues for a large body of research not just limited to perfect security, but also in other settings of statistical and computational security.

Bibliography

- [1] Aws latency monitoring. URL <https://www.cloudping.co/grid>. accessed February, 2022. [21](#)
- [2] Ittai Abraham and Kartik Nayak. Crusader agreement with $\leq 1/3$ error is impossible for $n \leq 3f$ if the adversary can simulate. *Decentralized Thoughts, Blog Post*, 2021. <https://tinyurl.com/decentralizedthoughts>, accessed: September 2021. [6](#)
- [3] Ittai Abraham, Danny Dolev, and Joseph Y Halpern. An almost-surely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 405–414, 2008. [79](#), [245](#), [250](#)
- [4] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $O(1)$ rounds, expected $o(n^2)$ communication, and optimal resilience. In *Financial Cryptography and Data Security, 2019*, volume 11598, pages 320–334. Springer, 2019. [22](#)
- [5] Ittai Abraham, Danny Dolev, and Gilad Stern. Revisiting asynchronous fault tolerant computation with optimal resilience. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 139–148, 2020. [11](#), [228](#)
- [6] Ittai Abraham, Gilad Asharov, and Avishay Yanai. Efficient perfectly secure computation with optimal resilience. In *Theory of Cryptography Conference*, 2021. [8](#), [23](#), [25](#), [33](#), [37](#), [75](#), [76](#), [77](#), [128](#)
- [7] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Asymptotically free broadcast in constant expected time via packed vss. In *Theory of Cryptography Conference*, 2022. [8](#), [75](#), [76](#), [77](#), [78](#), [79](#), [81](#), [92](#), [162](#), [186](#)

BIBLIOGRAPHY

- [8] Ittai Abraham, Danny Dolev, and Gilad Stern. Revisiting asynchronous fault tolerant computation with optimal resilience. *Distributed Computing*, 35(4):333–355, 2022. [9](#), [167](#)
- [9] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Detect, pack and batch: Perfectly-secure mpc with linear communication and constant expected time. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 251–281. Springer, 2023. [9](#), [167](#), [169](#), [173](#)
- [10] Ittai Abraham, Gilad Asharov, Arpita Patra, and Gilad Stern. Perfectly secure asynchronous agreement on a core set in constant expected time. *IACR Cryptol. ePrint Arch.*, page 1130, 2023. URL <https://eprint.iacr.org/2023/1130>. [179](#)
- [11] C Anirudh, Ashish Choudhury, and Arpita Patra. A survey on perfectly-secure verifiable secret-sharing. *ACM Computing Surveys*, 54(11s):1–36, January 2022. ISSN 1557-7341. [37](#), [80](#)
- [12] Ananya Appan and Ashish Choudhury. Network agnostic mpc with statistical security. In *Theory of Cryptography Conference*, pages 63–93. Springer, 2023. [231](#)
- [13] Ananya Appan, Anirudh Chandramouli, and Ashish Choudhury. Perfectly-secure synchronous mpc with asynchronous fallback guarantees. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 92–102, 2022. [2](#), [10](#), [11](#), [228](#), [231](#), [232](#), [233](#), [234](#), [238](#), [241](#), [242](#), [244](#), [245](#), [247](#), [248](#), [249](#), [250](#), [252](#), [255](#), [268](#)
- [14] Ananya Appan, Anirudh Chandramouli, and Ashish Choudhury. Network agnostic perfectly secure mpc against general adversaries. In *37th International Symposium on Distributed Computing (DISC 2023)*, 2023. [231](#)
- [15] Benny Applebaum and Eliran Kachlon. Conflict checkable and decodable codes and their applications. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1379–1424. SIAM, 2024. [168](#), [169](#), [177](#), [178](#)
- [16] Benny Applebaum, Eliran Kachlon, and Arpita Patra. The round complexity of perfect mpc with active security and optimal resiliency. In *Annual Symposium on Foundations of Computer Science (FOCS)*, 2020. [24](#)
- [17] Gilad Asharov and Anirudh Chandramouli. Optimal parallel broadcast. *Cryptology ePrint Archive*, 2024. [297](#)

BIBLIOGRAPHY

- [18] Gilad Asharov and Yehuda Lindell. A full proof of the bgw protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 2017. [13](#), [14](#), [32](#), [80](#)
- [19] Gilad Asharov, Yehuda Lindell, and Tal Rabin. Perfectly-secure multiplication for any $t < n/3$. In *Advances in Cryptology - CRYPTO 2011*, 2011. [8](#), [25](#), [75](#)
- [20] Gilad Asharov, Ran Cohen, and Oren Shochat. Static vs. adaptive security in perfect MPC: A separation and the adaptive security of BGW. In *3rd Conference on Information-Theoretic Cryptography, ITC 2022*, 2022. [iv](#), [5](#)
- [21] Laasya Bangalore, Ashish Choudhury, and Arpita Patra. Almost-surely terminating asynchronous byzantine agreement revisited. In *2018 ACM Symposium on Principles of Distributed Computing, PODC*. ACM, 2018. [79](#)
- [22] Laasya Bangalore, Ashish Choudhury, and Arpita Patra. The power of shunning: efficient asynchronous byzantine agreement revisited. *Journal of the ACM (JACM)*, 67(3): 1–59, 2020. [79](#), [245](#), [250](#)
- [23] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, 1991. [87](#), [161](#), [162](#), [178](#), [220](#), [222](#), [243](#), [244](#), [286](#), [291](#)
- [24] Zuzana Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 305–328. Springer, 2006. [78](#), [79](#)
- [25] Zuzana Beerliová-Trubíniová and Martin Hirt. Simple and efficient perfectly-secure asynchronous mpc. In *Advances in Cryptology – ASIACRYPT 2007*, pages 376–392, Berlin, Heidelberg, 2007. ISBN 978-3-540-76900-2. [10](#), [167](#)
- [26] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In *Proceedings of the 5th Conference on Theory of Cryptography, TCC'08*, page 213–230, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 354078523X. [8](#), [9](#), [26](#), [74](#), [76](#), [78](#), [167](#), [169](#)
- [27] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *Proc. of the Annual Symposium on Principles of Distributed Computing (PODC)*, 1983. [21](#), [27](#), [168](#)

BIBLIOGRAPHY

- [28] Michael Ben-Or and Ran El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16(4):249–262, 2003. [26](#)
- [29] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of Annual ACM Symposium on Theory of Computing*, 1988. [7](#), [8](#), [10](#), [21](#), [23](#), [24](#), [25](#), [32](#), [33](#), [74](#), [75](#), [78](#), [80](#), [82](#), [228](#), [253](#)
- [30] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of ACM symposium on Theory of computing*, 1993. [9](#), [10](#), [11](#), [38](#), [167](#), [170](#), [172](#), [228](#), [247](#), [253](#)
- [31] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '94, page 183–192, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916549. [9](#), [11](#), [167](#), [169](#), [228](#)
- [32] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *Advances in Cryptology—CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 663–680. Springer, 2012. [78](#), [79](#)
- [33] Piotr Berman, Juan A Garay, and Kenneth J Perry. Bit optimal distributed consensus. In *Computer science*. 1992. [6](#), [8](#), [21](#), [22](#), [23](#), [24](#), [26](#), [27](#), [75](#), [76](#), [79](#), [249](#)
- [34] Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In *Theory of Cryptography Conference*, pages 131–150. Springer, 2019. [2](#), [10](#), [231](#), [250](#)
- [35] Erica Blum, Jonathan Katz, and Julian Loss. Network-agnostic state machine replication. *arXiv preprint arXiv:2002.03437*, 2020. [231](#)
- [36] Erica Blum, Chen-Da Liu-Zhang, and Julian Loss. Always have a backup plan: fully secure synchronous mpc with asynchronous fallback. In *Annual International Cryptology Conference*, pages 707–731. Springer, 2020. [2](#), [10](#), [231](#), [232](#)
- [37] Gabriel Bracha. An asynchronous $[(n-1)/3]$ -resilient consensus protocol. In *PODC*, pages 154–162, 1984. [170](#), [173](#), [184](#), [225](#), [248](#), [249](#)

BIBLIOGRAPHY

- [38] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2): 130–143, 1987. [217](#)
- [39] Ran Canetti. Asynchronous secure computation. *Technion - Computer Science Department - Technical Report*, CS0755, 1993. [36](#), [37](#), [185](#), [247](#)
- [40] Ran Canetti. *Studies in secure multiparty computation and applications*. PhD thesis, Citeseer, 1996. URL <https://www.wisdom.weizmann.ac.il/~oded/PSX/ran-phd.pdf>. [9](#), [38](#), [167](#), [172](#), [179](#), [184](#), [186](#), [218](#), [220](#), [221](#), [232](#), [239](#), [247](#), [251](#), [252](#)
- [41] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1):143–202, 2000. [13](#), [14](#), [26](#)
- [42] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001. [12](#), [13](#), [183](#)
- [43] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. On adaptive vs. non-adaptive security of multiparty protocols. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques*, 2001. [iv](#), [5](#)
- [44] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. Adaptive versus non-adaptive security of multi-party protocols. *Journal of Cryptology*, 2004. [12](#)
- [45] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In *Advances in Cryptology—CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II 35*, pages 3–22. Springer, 2015. [12](#), [14](#), [16](#), [183](#)
- [46] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, 1988. [7](#), [8](#), [25](#), [74](#), [75](#)
- [47] Jinyuan Chen. Optimal error-free multi-valued byzantine agreement. In *DISC*, 2021. [6](#), [21](#), [22](#), [23](#), [24](#), [27](#)
- [48] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In

BIBLIOGRAPHY

- 26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 383–395. IEEE Computer Society, 1985. [24](#), [77](#)
- [49] Ashish Choudhury. *Protocols for Reliable and Secure Message Transmission*. PhD thesis, Citeseer, 2010. [18](#), [91](#), [246](#)
- [50] Ashish Choudhury and Arpita Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Transactions on Information Theory*, 2016. [25](#), [33](#), [87](#), [88](#), [160](#), [169](#), [174](#), [178](#), [179](#), [180](#), [187](#), [220](#), [239](#), [240](#), [244](#)
- [51] Ashish Choudhury, Martin Hirt, and Arpita Patra. Asynchronous multiparty computation with linear communication complexity. In *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2013. [25](#), [33](#)
- [52] Ashish Choudhury, Martin Hirt, and Arpita Patra. Unconditionally secure asynchronous multiparty computation with linear communication complexity. In *DISC*, 2013. [169](#)
- [53] Brian A Coan and Jennifer L Welch. Modular construction of nearly optimal byzantine agreement protocols. In *ACM Symposium on Principles of distributed computing*, 1989. [6](#), [8](#), [21](#), [22](#), [23](#), [24](#), [26](#), [27](#), [75](#), [76](#), [79](#)
- [54] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. *J. Cryptol.*, 32(3):690–741, 2019. [26](#), [27](#), [67](#)
- [55] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *International conference on the Theory and Applications of Cryptographic Techniques*, pages 311–326. Springer, 1999. [79](#)
- [56] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, 2000. [8](#), [25](#), [75](#)
- [57] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*, 2007. [160](#), [219](#)

BIBLIOGRAPHY

- [58] Ivan Damgård and Nikolaj I Schwartzbach. Communication lower bounds for perfect maliciously secure mpc. *Cryptology ePrint Archive*, 2020. [169](#), [297](#)
- [59] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2010. [9](#)
- [60] Ivan Damgård, Bernardo David, Irene Giacomelli, and Jesper Buus Nielsen. Compact vss and efficient homomorphic uc commitments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 213–232. Springer, 2014. [24](#)
- [61] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *ACM CCS Conference on Computer and Communications Security*, 2021. [35](#)
- [62] Giovanni Deligios and Chen-Da Liu-Zhang. Synchronous perfectly secure message transmission with optimal asynchronous fallback guarantees. In *International Conference on Financial Cryptography and Data Security*, pages 77–93. Springer, 2023. [231](#)
- [63] Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In *Theory of Cryptography Conference*, pages 623–653. Springer, 2021. [10](#), [231](#)
- [64] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. In *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, Canada August 18-20, 1982*, pages 132–140. ACM, 1982. [6](#), [21](#)
- [65] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 1985. [27](#)
- [66] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, 1988. [7](#), [21](#), [25](#), [26](#), [28](#), [32](#), [35](#), [46](#), [79](#)
- [67] Paul Neil Feldman. *Optimal Algorithms for Byzantine Agreement*. PhD thesis, Massachusetts Institute of Technology, 1988. [24](#), [78](#)

BIBLIOGRAPHY

- [68] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 1997. [27](#), [46](#)
- [69] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 1982. [20](#), [27](#), [79](#)
- [70] Matthias Fitzi and Juan A. Garay. Efficient player-optimal protocols for strong and differential consensus. In *PODC*, 2003. [26](#), [73](#)
- [71] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 699–710. ACM, 1992. [24](#), [77](#)
- [72] Chaya Ganesh and Arpita Patra. Optimal extension protocols for byzantine broadcast and agreement. *Distributed Computing*, 34(1):59–77, 2021. [27](#)
- [73] Rosario Gennaro, Michael O Rabin, and Tal Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *ACM symposium on Principles of distributed computing*, 1998. [8](#), [25](#), [75](#)
- [74] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *ACM symposium on Theory of computing*, 2001. [32](#)
- [75] Oded Goldreich and Erez Petrank. The best of both worlds: Guaranteeing termination in fast randomized byzantine agreement protocols. *Inf. Process. Lett.*, 36(1):45–49, 1990. [27](#)
- [76] Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional mpc with guaranteed output delivery. In *Annual International Cryptology Conference*, 2019. [8](#), [26](#), [74](#), [76](#), [78](#), [169](#)
- [77] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority mpc. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II*, pages 618–646, 2020. [78](#), [79](#), [297](#)
- [78] Vipul Goyal, Chen-Da Liu-Zhang, and Yifan Song. Towards achieving asynchronous mpc with linear communication and optimal resilience. *Cryptology ePrint Archive*, 2024. [298](#)

BIBLIOGRAPHY

- [79] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 466–485. Springer, 2010. [27](#)
- [80] Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In *International conference on the theory and application of cryptology and information security*, 2000. [8](#), [26](#), [74](#), [78](#)
- [81] Xiaoyu Ji, Junru Li, and Yifan Song. Linear-communication asynchronous complete secret sharing with optimal resilience. *Cryptology ePrint Archive*, 2024. [298](#)
- [82] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *Annual International Cryptology Conference*, 2006. [7](#), [21](#), [22](#), [23](#), [26](#), [27](#), [28](#), [32](#), [35](#), [65](#), [67](#), [69](#), [79](#)
- [83] 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*, 2008. [24](#)
- [84] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, 2006. [iv](#), [5](#), [12](#), [13](#)
- [85] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 1982. [6](#), [7](#), [20](#), [27](#), [74](#)
- [86] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. Sequential composition of protocols without simultaneous termination. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing, PODC 2002, Monterey, California, USA, July 21-24, 2002*, pages 203–212. ACM, 2002. [26](#)
- [87] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977. [18](#), [91](#), [246](#)
- [88] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with $\frac{t}{n/3}$, $O(n^2)$ messages, and $O(1)$ expected time. *Journal of the ACM (JACM)*, 62(4):1–21, 2015. [245](#)

BIBLIOGRAPHY

- [89] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. Improved extension protocols for byzantine broadcast and agreement. In *34th International Symposium on Distributed Computing*, 2020. [7](#), [21](#), [22](#), [23](#), [27](#)
- [90] Arpita Patra. Error-free multi-valued broadcast and byzantine agreement with optimal communication complexity. In *International Conference On Principles Of Distributed Systems*, 2011. [22](#), [27](#)
- [91] Arpita Patra, Ashish Choudhary, Tal Rabin, and C Pandu Rangan. The round complexity of verifiable secret sharing revisited. In *Advances in Cryptology-CRYPTO 2009: 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 487–504. Springer, 2009. [231](#), [238](#)
- [92] Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. Communication efficient perfectly secure VSS and MPC in asynchronous networks with optimal resilience. In *Progress in Cryptology - AFRICACRYPT 2010*, volume 6055, pages 184–202. Springer, 2010. [10](#), [167](#), [168](#), [247](#)
- [93] Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous verifiable secret sharing and multiparty computation. *Journal of Cryptology*, 28(1):49–109, 2015. [10](#), [167](#)
- [94] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 1980. [6](#), [7](#), [10](#), [20](#), [23](#), [27](#), [74](#), [228](#)
- [95] B. Prabhu, K. Srinathan, and C. Pandu Rangan. Asynchronous unconditionally secure computation: An efficiency improvement. In *Progress in Cryptology — INDOCRYPT 2002*, pages 93–107, Berlin, Heidelberg, 2002. ISBN 978-3-540-36231-9. [10](#), [167](#)
- [96] M. O. Rabin. Randomized byzantine generals. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 1983. [21](#), [27](#)
- [97] Tal Rabin. Robust sharing of secrets when the dealer is honest or cheating. *Journal of the ACM (JACM)*, 41(6):1089–1109, 1994. [231](#), [238](#)
- [98] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of ACM Symposium on Theory of Computing*, 1989. [77](#), [79](#), [231](#), [238](#)

BIBLIOGRAPHY

- [99] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960. 18
- [100] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. Synchronous distributed key generation without broadcasts. *IACR Cryptol. ePrint Arch.*, page 1635, 2021. URL <https://eprint.iacr.org/2021/1635>. 22
- [101] K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In *INDOCRYPT 2000*, pages 117–129, 2000. ISBN 978-3-540-44495-4. 10, 167
- [102] Georgios Tsimos, Julian Loss, and Charalampos Papamanthou. Gossiping for communication-efficient broadcast. *Cryptology ePrint Archive*, 2020. 27
- [103] Russell Turpin and Brian A Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Information Processing Letters*, 18(2):73–76, 1984. 28