STUDIES ON VERIFIABLE SECRET SHARING, BYZANTINE AGREEMENT AND MULTIPARTY COMPUTATION

A THESIS

submitted by

ARPITA PATRA

for the award of the degree

of

DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS

MAY 2010

Thesis Certificate

This is to certify that the thesis entitled **Studies on Verifiable Secret Sharing, Byzantine Agreement and Multiparty Computation** submitted by **Arpita Patra** to the Indian Institute of Technology Madras, Chennai for the award of the Degree of Doctor of Philosophy is a record of bona-fide research work carried out by her under my supervision and guidance. The contents of this thesis have not been submitted to any other university or institute for the award of any degree or diploma.

Chennai 600036 Research Guide

Date: (Prof. C. Pandu Rangan)

To The Supreme Personality of Godhead Sri Krsna and to my parents

Acknowledgments

> "Freedom consists not in refusing to recognize anything above us, but in respecting something which is above us; for by respecting it, we raise ourselves to it, and, by our very acknowledgment, prove that we bear within ourselves what is higher, and are worthy to be on a level with it."

My first and foremost heart-felt gratitude goes to my wonderful supervisor Prof. C. Pandu Rangan who introduced me to the world of 'Cryptography'. Prior to anything else, I would like to confess that I am among those luckiest few students in the world who are bestowed to receive a supervisor in the form of a father. It is rare to find such person who has extended the student-teacher relationship to that of a father-child bond. Truly, for him every student is like his own child (I feel more blessed, for I was one of his pet children!). It is not only academics, but for every problem and worry in my life, I have got him by my side. Still it is beyond my thought that I have to leave IIT Madras and move to a place where he will not be there around me. I remember my entry to IIT Madras with my eyes full of dreams to conquer the world and with a mind full of enthusiasm, eagerness and ready to plunge into every new source of knowledge. I never new GOD is planning to gift me one of the most precious gifts of my life in the form of my advisor, a complete package of teacher, father, dream-maker, friend, philosopher and the list grows on... I will never forget about those philosophical conversations we had many a times. As I look back now, I see how those conversations boosted my spirits, changed many aspects of my life immensely, ignited the passion of learning, generating and disseminating knowledge more and more without expecting much of material gain. I must confess that he is the perfect dream-maker. Many foundation stones of my dream are laid my him. His expertise in taking out the best out of a student is beyond appreciation. His inspiration can lift any student to excellence. He is a great orator and teacher. His oratory skill is phenomenal and it is something every student who has got a chance to listen to him will cherish. I wish that I can carry on the legacy of him to the next generation. Needless to mention, he has a tremendous influence on my professional as well as personal development. His guidelines in my initial days was of immense help for me. His colorful and creative feedback on my writing style has transformed each one of my drafts into excellent shape. Furthermore, I thank him for providing an excellent research atmosphere at the Theoretical Computer Science (TCS) Lab, IIT Madras. It has been a delight working in TCS Lab, thanks to the lively ambience maintained by the past and present students of the lab. TCS lab remains to be matchless and peerless in Department of Computer Science, as far as the seriousness, graveness in research is concerned with. At the same time, it is the most hip and happening place with full of fun, frolic and fiesta. I enjoyed every bit of my life in this lab which have created many great researchers in the past.

Beside my advisor, I also would like to acknowledge the members of my doctoral committee namely, Prof. P. Sreenivasa Kumar, Dr. A. Thangaraj, Prof. R. Rama, Dr. V. Kamakoti and Dr. Narayanaswamy for their encouragement and invaluable suggestions during my doctoral committee meetings. I would also like to thank the present and previous head of the department, Prof. C. Siva Ram

Murthy and Prof. T. A. Golsalves, respectively for their kind help. My heart-felt thanks to Prof. Kamala Krithivasan who has helped me a lot many times when I needed. I would also like to thank Dr. Sukhendu Das (who was my advisor during MS at IIT Madras) for helping me out in every little problem without hesitation.

I would like to thank Dr. Tal Rabin, for her collaboration on our CRYPTO paper and for her insightful remarks on some of our other articles. She was instrumental in teaching me how to write real 'good' paper. Her critical feedbacks were really eye-openers for me. Here I would also like to acknowledge Dr. Jonathan Katz and Ranjit Kumaresan for their collaboration. By sincere thanks to Dr. Jonathan Katz for allowing me to interact with his group at University of Maryland and supporting my stay unconditionally.

I owe a lot to my senior Srinathan who is at present an assistant professor at IIIT Hyderabad. A day's out with him will surely enlighten a person both academically and spiritually. I feel unfortunate for joining TCS Lab after he left the place. But still I relished every bit of those discussion sessions that we had at IIIT Hyderabad and at TCS Lab (when he used visit TCS Lab).

I am thankful to the unknown reviewers who rejected my papers several times in some of the international conferences and journals. The comments that they provided helped to polish our articles in better shape. But the bigger and nobler cause of thanking them is that the rejections have equipped me with high level of patience and helped me a lot to exercise/implement my spiritual thoughts in practice.

Several other people from whom I draw lot of inspiration are Prof. Bimal Roy, Prof. Palash Sarkar, Prof. Rana Barua from ISI Kolkata. I must say that they are just superb gang of people. They have helped me in many odd situations which I will never forget in my life. Something that always touches my heart is their extremely hearty behavior towards me. I am greatly thankful to them (specially Prof. Bimal Roy) for creating many opportunities for me for giving invited talks. I will never forget the day of my marriage when Prof. Bimal Roy and Prof. Palash Sarkar came to our residence. It was a very unexpected as well as cherishable moment for me. My heart was completely inundated with joy and the experience is beyond my words. This list will remain incomplete without two other names, Prof. Dipanwita RoyChoudhury and Prof. Indranil DasGupta from IIT Kharagpur. Both of them are very nice people and they have extended their helping hand in many occasions.

My lab mates Esha, Chaya, Billy, Thiru, Sai, Shinku, Amjed, Nisha, Saki, Manila, Kishore, Sobin, Madhu, Preetha madam, Sharmila and Vivek deserve special acknowledgments. Special mention goes to my cute sisters (Esha and Chaya), my sweet brothers (Billy, Thiru, Sai and Shinku), the pair (Vivek, Sharmi) and their little cute son Svitin (whom I wish to have as my future student :)) for their nice company. I also acknowledge my ex-lab-mates KP, Siddharth, Meena, Harini, Balu, Aswin, Raghavendra, Naresh, Nainesh and Naveen (some of them are exceptionally calm, cool and placid at heart). Aswin deserves special thanks for his collaboration over several of our works and also for his fruitful discussions.

The gang of close friends, Rima, Tanmoy, Suranjana, Anuradha, Annesha, Soma and Soumi, have stood by me though the ups and downs of my life. Their association has made my stay in IITM a pleasurable and memorable experience. The outings with Tanmoy and Rima will be etched permanently in my memory.

I will also not forget about those endless gossips, laughs that we had at our lunch table. Cheers to you guys! Your friendship is truly beyond words of acknowledgement.

My friends during my B. Tech days, Debi, Mitu, Saptarshee, Oishee, Soumen, Deepak Shah, Deepak Pal, for having faith and confidence in me and for showing enthusiasm and curiosity over my work. I still cherish the fresh excitements of my B. Tech days in their companion.

My sincere thanks to my in-laws (Dr. Ajit Choudhury and Chhanda Choudhury), Salim mama and brother-in-low Rahul for their unconditional support. They have always encouraged me for my study and above all they have unconditionally supported all that I wanted to do in my life.

My sincere thanks to all of our department office staffs (Radhai Madam, Saradha Madam, Balu Sir, Prema madam and Murali) and academic section stuffs (Saraswati Madam and Raghu Sir) for heartily helping me whenever I required.

My sincere acknowledgments to Microsoft Research (MSR), Bangalore for their Ph.D. fellowship and financial assistance during this work. I also acknowledge IARCS (Indian Association for Research in Computing Science) for supporting my travel to international conferences many time. I thank the organizing committee of CRYPTO and PODC for supporting my travel for the respective conferences. My sincere thanks to Google Inc. for supporting my entire cost of travel for PODC 2008 and ICITS 2008, held at Toronto and Calgary, Canada. I am thankful to Prof. Rei Safavi-Naini for supporting my stay at University of Calgary twice during ICITS 2008 and PODC 2009.

The placid and serene ambience at IIT Madras has played a key role in helping me out to keep my mental equilibrium. Undoubtedly, IIT Madras possesses the best natural campus among all the IITs in India. It was such a soothing experience to see and (sometimes feed) the deers and monkeys all around us. Literally, we got to experience the thrill of living with wild-life. In short, IIT Madras campus is simply rocking!

Now I would like to thank five remarkable people who had, are having and will continue to have a tremendous influence on my life. My parents, Ahibhushan Patra and Sima Patra, for introducing me to the world and bestowing me with all the love of the world and mental support that I needed in every step and every sphere of my life. My mother, the perfect dream-maker is the source of immense inspiration to me. "All the education become meaningless and purposeless if you do not become a good human being in that process": this is the lesson she taught me. My father is the first teacher in my life who taught me "Learn to live by struggle. Struggle is the color of life". My dearest sisters, Swagata (Liza) and Ankita (Lona), for their endless love for me. There is nothing in the world that I do not share with them. Moreover, I have got two best listeners in them. We three are a rocking trio. I thank GOD a lot to bless me with such a beautiful adorable family. I, from the bottom of my heart, sincerely want to be the child of such a wonderful parents and sister of such innocent siblings in all my births. This acknowledgement list will remain incomplete without the mention of my beloved husband Ashish, who also happened to be my research colleague. I have found a true life partner as well as research partner in him. I feel like I am one of the chosen children of GOD who has not only blessed me with loving family but also with such a wonderful husband. My every moment of research and personal life has been a cherishing experience in his company.

My development till this stage would have been incomplete without his insightful suggestions, fruitful discussions and critical remarks. He is my best and worst critic. Regarding any critical research discussion, I am sure that I can face the world fearlessly after I face him. He is truly a pillar of support for me.

Above all, I thank the Supreme personality of Godhead Krsna who created the universe and gave the mankind the supreme knowledge in the form of **Bhagavad Gita**, the ultimate manual for mankind and the quintessence of cosmic law in the eternal words of the supreme Lord Sri Krsna. He also teaches us "niskama karmayoga" i.e. the art of performing prescribed duties without having any attachment to its fruit and without having any material expectations (that I am trying my best to implement in my life):

कर्मण्येवाधिकारस्ते मा फलेषु कदाचन ।
मा कर्मफलहेतुर्भूर् मा ते संगोऽस्त्वकर्मणि ॥ ४७ ॥

> "You have a right to perform your prescribed duty, but you are not entitled to the fruits of action. Never consider yourself the cause of the results of your activities, and never be attached to not doing your duty."

I am truly indebted to all of them who come to the weekly programme at IIT Madras from ISKCON (International Society for Krsna Consciousness) to deliver talks on various topics from Bhagavad Gita. Each and every talk helped me to get elevated in the spiritual ladder and also to progress in my journey to the supreme Godhead Krsna. I also thank all the devotees for their association.

Abstract

This dissertation deals with three most important as well as fundamental problems in secure distributed computing, namely Verifiable Secret Sharing (VSS), Byzantine Agreement (BA) and Multiparty Computation (MPC).

VSS is a two phase protocol (Sharing and Reconstruction) carried out among n parties in the presence of a centralized adversary who can corrupt up to t parties. Informally, the goal of the VSS protocol is to share a secret s, among the n parties during the sharing phase in a way that would later allow for a unique reconstruction of this secret in the reconstruction phase, while preserving the secrecy of s until the reconstruction phase. VSS is used as a key tool in MPC, BA and many other secure distributed computing problems. It can take many different forms, depending on the underlying network (synchronous or asynchronous), the nature (passive or active) and computing power (bounded or unbounded) of the adversary, type of security (cryptographic or information theoretic) etc. We study VSS in *information theoretic setting* over both *synchronous* as well as *asynchronous* network, considering an *active unbounded powerful* adversary. Our main contributions for VSS are:

- In synchronous network, we carry out in-depth investigation on the *round complexity* of VSS by allowing a probability of error in computation and show that existing lower bounds for the round complexity of *error-free* VSS can be circumvented by introducing a negligible probability of error.

- We study the communication and round efficiency of VSS in synchronous network and present a robust VSS protocol that is simultaneously *communication efficient and round efficient*. In addition, our protocol is the best known communication and round efficient protocol in the literature.

- In asynchronous network, we study the communication complexity of VSS and propose a number of VSS protocols. Our protocols are highly communication efficient and show significant improvement over the existing protocols in terms of communication complexity.

The next problem that we deal with is Byzantine Agreement (BA). BA is considered as one of the most fundamental primitives for fault tolerant distributed computing and cryptographic protocols. BA among a set of n parties, each having a private input value, allows them to reach agreement on a common value even if some of the malicious parties (at most t) try to prevent agreement among the parties. Similar to the case of VSS, several models for BA have been proposed during the last three decades, considering various aspects like the underlying network, the nature and computing power of adversary, type of security. One of these models is BA over asynchronous network which is considered to be more realistic network than synchronous in many occasions. Though important, research in BA in asynchronous network has received much less attention in comparison to the BA protocols in synchronous network. Even the existing protocols for asynchronous BA involve high communication complexity and in general are very inefficient in comparison to their synchronous counterparts. We focus on BA in *information theoretic setting* over *asynchronous* network tolerating an *active* adversary having *unbounded computing power* and mainly work

towards the communication efficiency of the problem. Our contributions for BA are as follows:

- We propose communication efficient asynchronous BA protocols that show huge improvement over the existing protocols in the same setting. Our protocols for asynchronous BA use our VSS protocols in asynchronous network as their vital building blocks.

- We also construct a *communication optimal* asynchronous BA protocol for sufficiently long message size. Precisely, our asynchronous BA communicates $\mathcal{O}(\ell n)$ bits for ℓ bit message, for sufficiently large ℓ.

The studies on VSS and BA naturally lead one towards MPC problems. The MPC can model almost any known cryptographic application and uses VSS as well as BA as building blocks. MPC enables a set of n mutually distrusting parties to compute some function of their private inputs, such that the privacy of the inputs of the honest parties is guaranteed (except for what can be derived from the function output) even in the presence of an adversary corrupting up to t of the parties and making them misbehave arbitrarily. Much like VSS and BA, MPC can also be studied in various models. Here, we attempt to solve MPC in *information theoretic setting* over *synchronous* as well as *asynchronous* network, tolerating an *active unbounded powerful* adversary. As for MPC, our main contributions are:

- Using one of our synchronous VSS protocol, we design a synchronous MPC that minimizes the communication and round complexity simultaneously, where existing MPC protocols try to minimize one complexity measure at a time (i.e the existing protocols minimize either communication complexity or round complexity).

- We study the communication complexity of asynchronous MPC protocols and design a number of protocols for the same that show significant gain in communication complexity in comparison to the existing asynchronous MPC protocols.

- We also study a specific instance of MPC problem called Multiparty Set Intersection (MPSI) and provide protocols for the same.

In brief, our work in this thesis has made significant advancement in the state-of-the-art research on VSS, BA and MPC by presenting several inherent lower bounds and efficient/optimal solutions for the problems in terms of their key parameters such as communication complexity and time/round complexity. Thus our work has made a significant contribution to the field of secure distributed computing by carrying out a foundation research on the three most important problems of this field.

Contents

1 Introduction **1**

1.1 Overview of VSS, BA and MPC 2

1.1.1 Verifiable Secret Sharing (VSS) 2

1.1.2 Byzantine Agreement (BA) 3

1.1.3 Multiparty Computation (MPC) 3

1.2 Various Models for Studying VSS, BA and MPC 4

1.2.1 Communication Model 5

1.2.2 Adversary Model 6

1.3 The Model of our Interest and Informal Definitions of the Problems 8

1.4 History of Extant Literature on VSS, BA and MPC 9

1.4.1 The History of VSS in Synchronous Network 10

1.4.2 The History of VSS in Asynchronous Network 11

1.4.3 The History of MPC in Synchronous Network 12

1.4.4 The History of MPC in Asynchronous Network 12

1.4.5 The History of BA in Asynchronous Network 13

1.5 The Contribution of this Thesis 14

1.5.1 Investigation on The Round Complexity of Statistical VSS 14

1.5.2 Study of Communication and Round Efficiency of Statistical VSS 15

1.5.3 Study of Communication Efficiency of Statistical AVSS . . 16

1.5.4 Study of Communication Efficiency of Perfect AVSS 16

1.5.5 Study of Communication and Round Efficiency of Statistical MPC 16

1.5.6 Designing Efficient Multiparty Set Intersection Protocol in Synchronous Network 17

1.5.7 Study of Communication Efficiency of Statistical AMPC . 17

1.5.8 Study of Communication Efficiency of Perfect AMPC . . . 18

1.5.9 Designing Communication Efficient ABA for Small Message 18

1.5.10 Designing Communication Optimal ABA for Long Message 19

1.6 The Organization of this Thesis 20

I Results in Synchronous Network **24**

2 An Efficient Information Checking Protocol **25**

2.1 Introduction 25

2.1.1 Existing Literature and Existing Definition of ICP 25

2.1.2 New Definition, Model, Structure and Properties of ICP . 25

2.1.3 The Road-map 27

2.2 A Novel ICP 27

2.3 Comparison of MVMS-ICP with the ICPs of [138] and [48] 32
2.4 Some Important Remarks, Facts, Definitions and Notations 33
2.4.1 MVMS-ICP with One Round of Reveal 33
2.4.2 MVMS-ICP with Single Secret and $n = 3t + 1$ Verifiers . . 33
2.4.3 A Definition 34
2.4.4 Notation for using MVMS-ICP 34
2.5 Linearity of Protocol MVMS-ICP 34
2.6 Conclusion and Open Problems 37

3 The Round Complexity of Statistical VSS and WSS 38
3.1 Introduction 38
3.1.1 Relevant Literature of VSS 38
3.1.2 Our Results on Statistical VSS 39
3.1.3 Our Results on Statistical WSS 40
3.1.4 The Working Field of Our Protocols 41
3.1.5 On the Definition of Round Complexity of VSS and WSS . 41
3.1.6 The Network and Adversary Model 42
3.1.7 Definitions of VSS and WSS 42
3.1.8 The Road-map 43
3.2 Efficient 1-round Sharing, 2-round Reconstruction $(4, 1)$ Statistical VSS 44
3.2.1 Statistical VSS with One Round of Reconstruction 46
3.2.2 Statistical VSS with No Broadcast 46
3.3 Efficient 2-round Sharing, 2-round Reconstruction $(3t + 1, t)$ Statistical WSS 47
3.3.1 Statistical WSS with One Round of Reconstruction 51
3.3.2 Statistical WSS with One Round of Broadcast 51
3.4 Efficient 2-round Sharing, 2-round Reconstruction $(3t + 1, t)$ Statistical VSS 51
3.4.1 Statistical VSS with One Round of Reconstruction 55
3.4.2 Statistical VSS with One Round of Broadcast 55
3.5 Efficient 3-round Sharing, 2-round Reconstruction $(3, 1)$ Statistical VSS 55
3.5.1 3-round Sharing VSS with One Round of Reconstruction . 57
3.6 In-efficient 4-round Sharing, 2-round Reconstruction $(2t+1, t)$ Statistical VSS 57
3.6.1 4-round Sharing VSS with One Round of Reconstruction . 61
3.7 Efficient 5-round Sharing, 2-round Reconstruction $(2t + 1, t)$ Statistical VSS 61
3.7.1 5-round Sharing VSS with One Round of Reconstruction . 61
3.8 Lower Bounds for Statistical VSS 62
3.8.1 Lower Bound for 2-round Sharing Statistical VSS 62
3.8.2 Lower Bound for 1-round Sharing Statistical VSS 67
3.9 Efficient 1-round Sharing, 2-round Reconstruction $(3t + 1, t)$ Statistical WSS 70
3.9.1 1-round Sharing WSS with One Round of Reconstruction . 73
3.10 Efficient 3-round Sharing, 2-round Reconstruction $(2t + 1, t)$ Statistical WSS 74
3.10.1 3-round Sharing WSS with One Round of Reconstruction . 76
3.11 Lower Bounds for Statistical WSS 76

3.12 Conclusion and Open Problems 76

4 Communication and Round Efficient Statistical VSS 78
4.1 Introduction 78
4.1.1 Relevant Literature on Statistical VSS 78
4.1.2 Our Network and Adversary Model 78
4.1.3 Contribution of This Chapter 78
4.1.4 The Road-map 80
4.2 Statistical VSS For a Single Secret 80
4.2.1 The Output Generated by 5VSS-Share 85
4.2.2 Linearity Property of $1d^\star$-sharing and $2d^\star$-sharing 86
4.3 Statistical VSS For Multiple Secrets 87
4.3.1 The Output Generated by 5VSS-MS-Share 88
4.3.2 Linearity Property of $1d^{(\star,\ell)}$-sharing and $2d^{(\star,\ell)}$-sharing . . 91
4.4 Conclusion and Open Problems 93

5 Statistical MPC with Optimal Resilience Minimizing both Round and Communication Complexity 94
5.1 Introduction 95
5.1.1 Definition of MPC 95
5.1.2 The Relevant Literature on MPC 95
5.1.3 Statistical MPC with Optimal Resilience 95
5.1.4 Our Motivation and Contribution 98
5.1.5 Our Network and Adversary Model 99
5.1.6 The Road-map 100
5.2 Overview of Our Statistical MPC Protocol 100
5.3 Preparation Phase 100
5.3.1 Multiplication Protocol With Robust Fault Handling . . . 100
5.3.2 Conversion From $2d^{(\star,\ell)}$-sharing to ℓ Individual $2d^\star$-sharing 111
5.3.3 Preparation Phase — Main Protocol 113
5.4 Input Phase 114
5.5 Computation Phase 115
5.6 Statistical MPC Protocol 117
5.7 Conclusion and Open Problems 117

6 Statistical Multiparty Set Intersection 118
6.1 Introduction 118
6.1.1 Secure Multiparty Set Intersection (MPSI) 118
6.1.2 Existing Literature on MPSI 119
6.1.3 The Network and Adversary Model 120
6.1.4 Our Motivation and Contribution 120
6.1.5 The Road-map 121
6.2 Round and Communication Complexity of MPSI Protocol of [116] 121
6.3 Discussion on Our New MPSI Protocol with $n = 3t + 1$ 123
6.3.1 Our MPSI Protocol with $n = 3t + 1$ vs. Existing General MPC Protocols 123
6.3.2 The Working Field of our MPSI Protocol 124
6.3.3 Overview of Our Protocol 124
6.4 Generation of a Random Value 125
6.5 Statistical VSS with $n = 3t + 1$ 125

6.5.1 The Output Generated by VSS-Share 130
6.6 Generating Random t-$(1d)$-sharing 131
6.7 Multiplication Protocol 132
6.7.1 Upgrading t-$(1d)$-sharing to t-$(2d)$-sharing 132
6.7.2 An ABC protocol— Proving $c = ab$ 135
6.7.3 Our Multiplication Protocol 138
6.8 Statistical MPSI Protocol with $n = 3t + 1$ 139
6.9 Statistical MPSI Protocol with Optimal Resilience 146
6.9.1 Preparation Phase 146
6.9.2 Input Phase 147
6.9.3 Computation and Output Phase 147
6.9.4 Our New MPSI with Optimal Resilience 148
6.9.5 Our MPSI Protocol with $n = 2t + 1$ vs. Existing General MPC Protocols 149
6.10 Conclusion and Open Problems 149

II Results in Asynchronous Network 151

7 Efficient Asynchronous Information Checking Protocols 152
7.1 Introduction 152
7.1.1 Existing Literature and Definition of Asynchronous ICP or AICP 152
7.1.2 New Definition, Model, Structure and Properties of AICP 152
7.2 A-cast: Asynchronous Broadcast 155
7.3 Our First AICP 156
7.4 Our Second AICP 159
7.5 Discussion About MVMS-AICP-I and MVMS-AICP-II 165
7.6 Comparison of MVMS-AICP-I and MVMS-AICP-II with Existing AICP of [39, 35] 165
7.7 Definition and Notations for Using MVMS-AICP-I and MVMS-AICP-II as Black Box 165
7.8 Conclusion and Open Problems 166

8 Efficient Statistical AVSS Protocols With Optimal Resilience 168
8.1 Introduction 168
8.1.1 The Network and Adversary Model 169
8.1.2 Definitions 169
8.1.3 Contribution of This Chapter 171
8.1.4 The Road-map 172
8.2 Discussion on the Approaches used in the AVSS of [39] and the Approaches used by our AVSS Protocols 172
8.3 Statistical AWSS Protocol 173
8.3.1 AWSS Scheme for Sharing a Single Secret 173
8.3.2 AWSS Scheme for Sharing Multiple Secrets 180
8.3.3 Deriving Two AWSS Protocols for Single Secret from Protocol AWSS 183
8.3.4 Deriving Two AWSS Protocols for Multiple Secrets from Protocol AWSS-MS 184
8.4 Our Weak Statistical AVSS protocol 186

8.4.1 Our Weak Statistical AVSS Scheme for Sharing a Single Secret . . . 186
8.4.2 Deciding The Choice of AWSS Protocol . . . 191
8.4.3 Our Weak Statistical AVSS Scheme for Sharing Multiple Secrets . . . 192
8.4.4 Deciding The Choice of AWSS Protocol . . . 194
8.5 Our Strong Statistical AVSS protocol . . . 197
8.5.1 Our Strong Statistical AVSS Scheme for Sharing a Single Secret . . . 197
8.5.2 Deciding The Choice of AWSS Protocol . . . 207
8.5.3 Our Strong Statistical AVSS Scheme for Sharing Multiple Secrets . . . 208
8.5.4 Deciding The Choice of AWSS Protocol . . . 209
8.6 Conclusion and Open Questions . . . 212
8.7 Appendix: Analysis of the Communication Complexity of the AVSS Scheme of [39] . . . 213

9 Efficient ABA with Optimal Resilience for Short Message 215
9.1 Introduction . . . 215
9.1.1 The Network and Adversary Model . . . 216
9.1.2 Definitions . . . 216
9.1.3 Relevant History of ABA . . . 217
9.1.4 The Motivation of Our Work . . . 219
9.1.5 Contribution of This Chapter . . . 219
9.1.6 The Road-map . . . 221
9.2 A Brief Discussion on the Approaches Used in the ABA Protocols of [39, 1] and Current Chapter . . . 221
9.3 Our ABA Protocol for Single Bit . . . 222
9.3.1 Existing Common Coin Protocol Using Our AVSS Protocol 222
9.3.2 Existing Voting Protocol . . . 227
9.3.3 The ABA Protocol for Single Bit . . . 230
9.4 Our Efficient ABA Protocol for Multiple Bits . . . 233
9.4.1 An Incorrect Common Coin Protocol . . . 234
9.4.2 A New and Efficient Common Coin Protocol for Multiple Bits . . . 236
9.4.3 Final ABA Protocol for Achieving Agreement on $t+1$ Bits Concurrently . . . 243
9.5 Conclusion and Open Problems . . . 246
9.6 APPENDIX: Analysis of the Communication Complexity of the ABA Scheme of [39, 35] . . . 247

10 Efficient Statistical AMPC with Optimal Resilience 248
10.1 Introduction . . . 248
10.1.1 Network and Adversary Model . . . 248
10.1.2 Definitions . . . 248
10.1.3 Relevant History of Statistical Asynchronous MPC . . . 250
10.1.4 Contribution of This Chapter . . . 250
10.1.5 The Road-map . . . 251
10.2 Statistical ACSS . . . 251
10.2.1 Tool Used for our Statistical ACSS . . . 251

10.2.2 Statistical ACSS for Sharing a Single Secret 251
10.2.3 Statistical ACSS for Sharing Multiple Secrets 256
10.3 Primitives Used in Our AMPC Protocol 259
10.4 The Approach Used in the AMPC of [21] and Current Chapter . . 259
10.5 Generating t-$(2d)$-Sharing 261
10.6 Preparation Phase 264
10.6.1 Generating Secret Random t-$(2d)$-sharing 264
10.6.2 An ABC Protocol– Proving $c = ab$ 267
10.6.3 Generating Multiplication Triples: The Main Protocol For Preparation Phase 269
10.7 Input Phase 271
10.8 Computation Phase 272
10.9 The New Statistical AMPC Protocol with Optimal Resilience . . 274
10.10 Conclusion and Open Problems 275

11 Efficient Statistical AVSS Protocol With Non-Optimal Resilience and Perfect AVSS With Optimal Resilience 276
11.1 Introduction 277
11.1.1 The Network and Adversary Model 277
11.1.2 The Definitions 277
11.1.3 Relevant Literature 278
11.1.4 Contribution of This Chapter 278
11.1.5 The Motivation for Presenting our AVSS Schemes 279
11.1.6 The Common Primitives Used for Both of our AVSS Schemes 280
11.1.7 The Road-map 280
11.2 Statistical AVSS For Sharing a Single Secret 281
11.2.1 **Distribution** Phase 281
11.2.2 **Verification & Agreement on CORE** Phase 282
11.2.3 **Generation of τ-$(1d)$-sharing** Phase 289
11.2.4 Protocol St-AVSS: Statistical AVSS Sharing a Single Secret 290
11.3 Statistical AVSS For Sharing Multiple Secrets 292
11.4 A Different Interpretation of Protocol St-AVSS-MS 293
11.5 Finding (n,t)-star Structure in a Graph 295
11.6 Perfect AVSS for Sharing a Single Secret 298
11.6.1 Distribution Phase 299
11.6.2 Verification & Agreement on CORE Phase 299
11.6.3 Generation of τ-$(1d)$-sharing Phase 303
11.6.4 Protocol Pf-AVSS-Share and Pf-AVSS-Rec 304
11.7 Perfect AVSS for Sharing Multiple Secrets 306
11.8 A Different Interpretation of Protocol Pf-AVSS-MS 308
11.9 Conclusion and Open Problems 309

12 Efficient Statistical AMPC Protocol With Non-Optimal Resilience and Perfect AMPC With Optimal Resilience 310
12.1 Introduction 311
12.1.1 The Network and Adversary Model 311
12.1.2 Definitions 311
12.1.3 Relevant Literature on AMPC 311
12.1.4 Contribution of This Chapter 313
12.1.5 Primitives Used 313

12.1.6 The Road-map 315
12.2 Statistical AMPC of Huang et al. [107] 315
12.3 Statistical Protocol for Generating $(t, 2t)$-$(1d)$-sharing of ℓ Secrets 317
12.4 Statistical AMPC Protocol with $n = 4t + 1$ 319
12.4.1 Preparation Phase 319
12.4.2 Input Phase 322
12.4.3 Computation Phase 324
12.4.4 Our Statistical AMPC Protocol 325
12.5 Perfect Protocol for Generating $(t, 2t)$-$(1d)$-sharing of ℓ Secrets . . 326
12.5.1 Comparison with Existing Protocol for generating $(t, 2t)$-$(1d)$-sharing 326
12.6 Our Perfect AMPC Protocol Overview 327
12.6.1 Preparation Phase 327
12.6.2 Input Phase 328
12.6.3 Computation Phase 328
12.6.4 Our Perfect AMPC Protocol 328
12.7 Conclusion and Open Problems 329

13 Efficient Statistical ABA Protocol With Non-Optimal Resilience 330
13.1 Introduction 331
13.1.1 The Network and Adversary Model 331
13.1.2 Our Motivation and Contribution 331
13.2 Our ABA protocol with Non-optimal Resilience 332
13.2.1 A New and Efficient Common Coin Protocol for Multiple Bits with $n = 4t + 1$ 332
13.2.2 Final ABA Protocol for Achieving Agreement on $2t+1$ bits Concurrently with $n = 4t + 1$ 334
13.3 Conclusion 334

14 Communication Optimal Multi-Valued A-cast and ABA with Optimal Resilience 335
14.1 Introduction 336
14.1.1 The Network and Adversary Model 337
14.1.2 Definitions 337
14.1.3 The History of Asynchronous Broadcast or A-cast 338
14.1.4 The History of Asynchronous Byzantine Agreement (ABA) 338
14.1.5 Multi-valued A-cast and ABA: Motivation of Our work . . 339
14.1.6 Contribution of This Chapter 340
14.1.7 The Road-map 342
14.2 Communication Optimal (ϵ, δ)-A-cast Protocol 342
14.2.1 Tools Used 342
14.2.2 Protocol Optimal-A-cast 343
14.3 Communication Optimal (ϵ, δ)-ABA Protocol 351
14.3.1 Tools Used 351
14.3.2 Approach used in the BA protocol of [75] 353
14.3.3 Protocol Optimal-ABA 354
14.4 Conclusion and Open Problem 370

III Summary, Discussions and Future Directions **371**

15 Conclusion **372**
15.1 Summary of Contributions 372
15.2 Insightful Inferences 373
15.3 Future Works and Future Directions 375
15.3.1 Future Work of Type I 375
15.3.2 Future Work of Type II 376

List of Figures

2.1 Protocol MVMS-ICP with $n = 2t + 1$ Verifiers 29
2.2 Linearity of Protocol MVMS-ICP Over Addition Operation. 36

3.1 1-Round Sharing, 2-Round Reconstruction $(4, 1)$ Statistical VSS. . 45
3.2 2-Round Sharing, 2-Round Reconstruction $(3t + 1, t)$ Statistical WSS. 48
3.3 2-Round Sharing, 2-Round Reconstruction $(3t+1, t)$ Statistical VSS. 53
3.4 A 3-Round Sharing 2-Round Reconstruction $(3, 1)$ Statistical VSS protocol. 56
3.5 Sharing Phase of 4-round sharing 2-round reconstruction $(2t+1, t)$ statistical VSS. 58
3.6 Reconstruction Phase of 4-round sharing 2-round reconstruction $(2t + 1, t)$ statistical VSS. 59
3.7 A 1-Round Sharing 2-Round Reconstruction $(3t + 1, t)$ Statistical WSS 72
3.8 A 3-Round Sharing 2-Round Reconstruction $(2t + 1, t)$ Statistical WSS 75

4.1 Sharing Phase of 5-Round Sharing, 2-Round Reconstruction $(2t + 1, t)$ Statistical VSS 81
4.2 Reconstruction Phase of 5-round sharing 2-round reconstruction $(2t + 1, t)$ statistical VSS 82
4.3 Sharing Phase of $(2t + 1, t)$ statistical VSS Scheme 5VSS-MS . . . 89
4.4 Reconstruction Phase of $(2t + 1, t)$ statistical VSS Scheme 5VSS-MS 90

5.1 Protocol for Generating $2d^{(\star,\ell)}$-sharing of ℓ random values. 101
5.2 Public Reconstruction of ℓ Values that are $1d^{(\star,\ell)}$-shared by some party P. 103
5.3 Protocol to Generate $2d^{(\star,\ell)}$-sharing of $(c^1, \ldots, c^\ell)$ where $c^l = a^l b^l$ for $l = 1, \ldots, \ell$. 106
5.4 Robust Multiplication Protocol. 109
5.5 Protocol for converting $2d^{(\star,\ell)}$-sharing to ℓ separate $2d^\star$-sharing. . 112
5.6 Protocol for generating $2d^\star$-sharing of $c_M + c_R$ random multiplication triples $((a^l, b^l, c^l)\ ;\ l = 1, \ldots, c_M + c_R)$. 114
5.7 Protocol for generating $2d^\star$-sharing of the inputs of each party. . . 115
5.8 Protocol for computing the circuit. 116

6.1 Protocol RandomVector: Generates a random value. 126
6.2 Protocol VSS-Share: Sharing Phase of Protocol VSS. 127
6.3 Protocol VSS-Rec: Reconstruction Phase of Protocol VSS 127
6.4 Protocol Random: Generates t-$(1d)$-sharing of ℓ random secrets. . 132

6.5 Protocol Upgrade1dto2d: Generates t-$(2d)$-sharing of ℓ secrets given t-$(1d)$-sharing of the same secrets. 134
6.6 Protocol ProveCeqAB: An ABC Protocol for proving $c = ab$. . . . 136
6.7 Protocol Mult: Generates $[c^l]_t$ from $[a^l]_t$ and $[b^l]_t$ for $l = 1, \ldots, \ell$. . 139
6.8 Input and Preparation Phase of our statistical MPSI Protocol . . 140
6.9 Protocol for Computation Phase and Output Phase of our MPSI protocol. 142
6.10 Computation Phase and Output phase of our Statistical MPSI Protocol 148

7.1 Bracha's A-cast Protocol with $n = 3t + 1$ 155
7.2 Our First AICP with $n = 3t + 1$ Verifiers. 157
7.3 Our First AICP with $n = 3t + 1$ Verifiers. 158
7.4 Our second AICP with $n = 3t + 1$ 160
7.5 Our second AICP with $n = 3t + 1$ 161

8.1 Sharing Phase of Protocol AWSS for single secret s with $n = 3t + 1$ 175
8.2 Reconstruction Phase of AWSS Scheme for single secret s with $n = 3t + 1$ 177
8.3 Sharing Phase of Protocol AWSS-MS for Sharing S Containing $\ell \geq 1$ Secrets 181
8.4 Reconstruction Phases of AWSS-MS for Sharing S Containing ℓ Secrets 182
8.5 Sharing Phase of our Weak Statistical AVSS Scheme for Sharing a Single Secret s with $n = 3t + 1$ 188
8.6 Reconstruction Phase of our Weak Statistical AVSS Scheme for Sharing a Single Secret s with $n = 3t + 1$ 189
8.7 Sharing Phase of Weak Statistical AVSS Scheme for Sharing a Secret S Containing ℓ Elements 194
8.8 Reconstruction Phase of our Weak Statistical AVSS Scheme for Sharing Secret S Containing ℓ Elements. 195
8.9 Code for Commitment by D Phase 199
8.10 Code for Verification of D's Commitment Phase 201
8.11 Code for "Re-commitment by Individual Parties" Phase 204
8.12 Our Strong Statistical AVSS for Sharing Secret s with $n = 3t + 1$ 205
8.13 Code for Commitment by D Phase for $\ell \geq 1$ secrets 209
8.14 Code for Verification of D's Commitment Phase for $\ell \geq 1$ secrets . 210
8.15 Code for "Re-commitment by Individual Parties" Phase for $\ell \geq 1$ secrets 210
8.16 Our Strong Statistical AVSS for Sharing $\ell \geq 1$ Secrets with $n = 3t + 1$ 211

9.1 Existing Common Coin Protocol 224
9.2 Existing Vote Protocol 229
9.3 Efficient ABA Protocol for Single Bit. 231
9.4 An Incorrect Common Coin protocol obtained by replacing WAVSS-Share and WAVSS-Rec-Public by WAVSS-MS-Share and WAVSS-MS-Rec-Public respectively in Protocol Common-Coin 235
9.5 Specific Adversary Behavior in Protocol Common-Coin-Wrong . . . 237
9.6 Multi-Bit Common Coin Protocol using Protocol WAVSS-MS-Share and WAVSS-MS-Rec-Public as Black-Boxes 240

9.7 ABA Protocol to Reach Agreement on $n - 2t = t + 1$ Bits 245

10.1 Protocol ACSS-Share for Sharing Secret s with $n = 3t + 1$ 253
10.2 Protocol ACSS-Rec-Private and ACSS-Rec-Public for Reconstructing Secret s privately and publicly (respectively) with $n = 3t + 1$. 254
10.3 Protocol ACSS-MS-Share for Sharing Secret S Containing ℓ Elements with $n = 3t + 1$ 257
10.4 Protocol ACSS-MS-Rec-Private and ACSS-MS-Rec-Public for Reconstructing Secret S privately and publicly (respectively) with $n = 3t + 1$ 258
10.5 Protocol t-(2d)-Share for Generating t-$(2d)$-sharing of $S = (s^1, \ldots, s^\ell)$, $n = 3t + 1$ 263
10.6 Protocol for Collectively Generating t-$(2d)$-sharing of ℓ secrets, $n = 3t + 1$ 266
10.7 Protocol for Generating t-$(1d)$-sharing of $[c^1]_t = [a^1]_t.[b^1]_t, \ldots, [c^\ell]_t = [a^\ell]_t.[b^\ell]_t$, $n = 3t + 1$ 268
10.8 Protocol for Generating t-$(1d)$-sharing of $c_M + c_R$ secret random multiple triples 270
10.9 Protocol for Input Phase, $n = 3t + 1$ 272
10.10 Protocol for Computation Phase (Evaluating the Circuit), $n = 3t+1$ 273

11.1 First Phase of Protocol St-AVSS-Share: **Distribution** Phase . . . 282
11.2 Steps to be executed with respect to a Single Verifier 284
11.3 Second Phase of Protocol St-AVSS-Share: **Verification & Agreement on CORE** phase 288
11.4 Third Phase of protocol St-AVSS-Share: **Generation of τ-$(1d)$-sharing** 289
11.5 Protocol St-AVSS 290
11.6 First Phase of Protocol St-AVSS-MS-Share: **Distribution** Phase . 293
11.7 Steps to be executed with respect to a Single Verifier for multiple secrets 294
11.8 Second Phase of Protocol St-AVSS-MS-Share: **Verification & Agreement on CORE** phase 295
11.9 Third Phase of protocol St-AVSS-MS-Share: **Generation of τ-$(1d)$-sharing** Phase 295
11.10 Protocol St-AVSS-MS 296
11.11 Algorithm For Finding (n, t)-star 297
11.12 First Phase of Protocol Pf-AVSS-Share: **Distribution by D** Phase 299
11.13 Second Phase of Protocol Pf-AVSS-Share: **Verification & Agreement on CORE** phase 302
11.14 Third Phase of protocol Pf-AVSS-Share: **Generation of τ-$(1d)$-sharing** 304
11.15 Perfect AVSS protocol: Pf-AVSS 305
11.16 First Phase of Protocol Pf-AVSS-MS-Share: **Distribution by D** Phase 306
11.17 Second Phase of Protocol Pf-AVSS-MS-Share: **Verification & Agreement on CORE** phase 307
11.18 Third Phase of protocol Pf-AVSS-MS-Share: **Generation of τ-$(1d)$-sharing** Phase 307
11.19 Our Perfect AVSS protocol: Protocol Pf-AVSS-MS 308

12.1 Protocol for Agreement on a Common Subset with $n = 4t + 1$ . . 314
12.2 Steps for Generating t-$(1d)$-sharing of Random a and b in a Segment in the BSS Scheme of Zheng et al. [152] 316
12.3 Protocol for Generating $(t, 2t)$-$(1d)$-sharing of ℓ secrets Concurrently. 318
12.4 Preparation Phase: Generation of $(t, 2t)$-$(1d)$-sharing of $c_M + c_R$ secret random values. 320
12.5 Input Phase: Generation of t-$(1d)$-sharing of the Inputs. 322
12.6 Computation Phase: Evaluation the Circuit. 325

13.1 Multi-Bit Common Coin Protocol using Protocol Pf-AVSS-MS-Share and Pf-AVSS-MS-Rec as Black-Boxes 333

14.1 Protocols for First Two Phases of Optimal-A-cast: **Distribution Phase** and **Verification & Agreement on CORE Phase** . . . 345
14.2 Protocol for Last Phase of Optimal-A-cast: **Output Phase** 347
14.3 Protocol Optimal-A-cast: Communication Optimal A-cast protocol. 350
14.4 Protocol for Agreement on a Common Subset with $n = 3t + 1$ . . 352
14.5 Overall structure of Protocol Optimal-ABA. 356
14.6 Code for Checking Phase. 358
14.7 Code for the Expansion Phase. 361
14.8 Code for **Output Phase** 365

List of Tables

1.1 The Attributes of Communication Model. 6
1.2 The Attributes of Adversary Model. 8

2.1 Communication Complexity and Round Complexity of protocol MVMS-ICP and Existing ICP with $n = 2t + 1$ verifiers and ℓ secrets. 33

3.1 Summary of VSS Bounds and Round Complexity. 39
3.2 Summary of Statistical VSS Bounds and Round Complexity. . . . 40

4.1 Communication Complexity and Round Complexity of our statistical VSS and Existing Statistical VSS Schemes with $n = 2t + 1$. 79

5.1 Communication Complexity and Round Complexity of Existing statistical MPC protocols with Optimal Resilience. 99

6.1 Comparison of our MPSI protocol with the MPSI protocol of [116]. 123
6.2 Comparison of our MPSI with the general MPC protocols that securely compute (6.1). 124
6.3 Comparison of our MPSI with the general MPC protocols that securely compute (6.1). 150

7.1 Communication Complexity of protocol MVMS-AICP-I, MVMS-AICP-II and Existing AICP of [39, 35] with $n = 3t + 1$ verifiers and ℓ secrets. 165

8.1 Comparison of our AVSS protocols with the exiting AVSS Protocol of [39, 35] in terms of Communication Complexity. 172

9.1 Summary of Best Known Existing ABA Protocols 218
9.2 Comparison of Our ABA with Best Known Optimally Resilient ABA Protocols 220

10.1 Existing Statistical AMPC Protocols. 250

11.1 Comparison of our AVSS protocols with Existing AVSS Protocols. CC: Communication Complexity 279

12.1 Communication complexity (CC) in bits per multiplication gate of known AMPC protocols. 312

14.1 Summary of Best Known Existing ABA Protocols 339
14.2 Our Contribution 341
14.3 Corresponding Black box Protocols and their Properties. 341

Chapter 1

Introduction

Cryptography, the science of secrecy, is the art of keeping a secret as secret. It is the study of secure information exchange in an insecure environment. Cryptographic applications have been explored for a few centuries and the earliest known cryptographic protocols dates back to the period of Julius Caesar who is known to invent and use *Caesar Cypher*. Historically, cryptography was exclusively concerned with securely communicating messages in the presence of an adversary. Till the first half of the previous century, the study and use of cryptography was confined to the domain of militaries and governments. But the tremendous and explosive growth of Internet in the past few decades has brought cryptography and security out of the realms of the powers into the public domain. Security concerns are inherent in any system that needs mutually unknown parties to interact among themselves over a distributed network. One such area of mutual interaction, that has captured the witty imaginations and insightful thoughts of a lot of mathematicians and researchers in the past three decades is the field of *secure distributed computing*. *Secure distributed computing* can model any cooperative computation, where people jointly conduct computation tasks based on the private inputs they each supplies. These computations could occur between mutually un-trusted parties, or even between competitors. For example, customers might send queries to a re-mote database that contain private information; two competing financial organizations might jointly invest in a project that must satisfy both organizations private and valuable constraints, and so on. The world of *secure distributed computing* more or less revolves around the following three mutually dependent, yet independently motivated, core problems:

1. Verifiable Secret Sharing (VSS);
2. Byzantine Agreement (BA);
3. Multiparty Computation (MPC).

This dissertation deals with these three fundamental and important problems and contributes significantly for advancement of the state-of-the-art research on these three problems. This chapter is now molded in the following manner: First we give an overview of the problems which starts by tracing the genesis of each of the individual problems and ends with the current standard interpretation of the problems. Then we list different models in which the problems have been studied so far and can be looked at in future. Next, we present the extant history on each of the problems. In this part we also consciously try to bring forth how these problems have grown interdependency among them, though created with different motivations. We then emphasize on our contributions in this thesis and their impacts on the literature. This will help to judge the stand that our results hold with respect to the past history and also to understand how our results have advanced the state-of-the-art research of this field. Lastly, we describe the chapter wise organization of this thesis.

1.1 Overview of VSS, BA and MPC

1.1.1 Verifiable Secret Sharing (VSS)

VSS finds its origin in one of the classical cryptographic problems called secret sharing [140, 27, 22]. Secret sharing deals with the techniques to share secrets among parties in such a way that only designated subset of parties can reconstruct the shared secret and no other subset of parties can reconstruct the secret. It finds extensive use in key management, distributed storage system etc. To be more precise, secret sharing is a two phase protocol (sharing, reconstruction) carried out among n parties. In the sharing phase a special party called *dealer* shares a secret s among the n parties in such a way that later any designated subset of parties (specified by access structure) can reconstruct the shared secret s uniquely and no other subset of parties (specified by adversary structure) can reconstruct s. Secret sharing has been classified in many types, e.g.

1. *Cryptographic* (the secrecy of the secret depends on the difficulty of solving certain number-theoretic hard problem) or *Information theoretic* [140, 27] (the secrecy of the secret is not dependent on the hardness of any computational problem).

2. *Threshold* [140, 27] (for a fixed threshold t, any set of $t+1$ parties can uniquely reconstruct the secret i.e. access structure is the set of all different combinations of $t+1$ parties) or *Non-threshold* [22] (generalization of threshold; Access structure may have sets of parties of different size).

3. *Static* [140, 27, 22] (the shares of secret remain the same after the distribution) or *Proactive/Mobile* [38, 97, 124] (the shares can be refreshed or redistributed without changing the secret in order to maintain secrecy over long periods).

Despite all these classifications, secret sharing can not withstand in real-life applications, for it makes an unrealistic assumption that all the parties behave honestly throughout in a system. That is, the parties in adversary structure may at most behave like a eavesdropper who can simply learn the information of other corrupted parties and try to obtain some information by manipulating the collected data. So the big question comes that what would happen if some of the parties stray away from their designated instructions to communicate/compute in any arbitrary fashion and collaborate among themselves in a centralized fashion to get some extra advantage. There are two main problems that may arise. In the sharing phase, the dealer may share no valid secret and get away with it. In the reconstruction phase, the bad/corrupted parties may input some wrong shares and prevent the reconstruction of secret. The above two problems clearly say that secret sharing is not equipped to tolerate malicious faults. To overcome this problem, the first effort came from Tompa and Woll [147] and McEliece and Sarwate [121], who gave some *partial* solutions considering faults in the system.

After that, the notion of VSS was introduced by Chor, Goldwasser, Micali and Awerbuch in [43] to completely resolve the concern.

Informally, a VSS is a two phase protocol (Sharing and Reconstruction) carried out among n parties in the presence of a malicious/active adversary (how the corruption is done depends on different model discussed later). The goal of the VSS protocol is to share a secret, s, among the n parties during the sharing phase in a way that would later allow for a unique reconstruction of this secret in the reconstruction phase, while preserving the secrecy of s until the reconstruction phase. In many applications one may treat VSS as a form of commitment where the commitment information is held in a distributed fashion by the parties. Most importantly, in the distributed setting the de-commitment is guaranteed, that is the committed value will be exposed. This is in contrast to the non-distributed setting where the committer can decide whether to expose the value or not.

After the original introduction of the concept of VSS in [43], VSS has emerged as one of the fundamental primitives in secure distributed computing and it finds lot of application in MPC, BA, threshold signature schemes, secret ballot elections and all other applications of secret sharing. After [43], many VSS protocols were proposed motivated by various applications. In [95, 20, 41, 138, 93, 48, 49], VSS protocols are devised as tool for MPC. In [64, 133], VSS protocols were devised for the task of sharing secrets of discrete-log based cryptosystems. In [67, 39, 35], VSS protocols are designed to be used in BA. In several other works [91, 73, 109], VSS is considered as a stand alone application for studying its round complexity.

1.1.2 Byzantine Agreement (BA)

The problem of BA (popularly known as Byzantine General's Problem) is a classical problem in distributed computing introduced by Lamport et al. in [115]. In many practical situations, it is necessary for a group of parties (or processes) in a distributed system to agree on some issue, despite the presence of some faulty parties who may try to make the honest parties disagree. BA is the primitive to solve the above mentioned problem. The most basic and commonly used form of BA is as follows: BA among a set of n parties each having a private input value, allows them to reach agreement on a common value even if some of the parties are faulty and try to prevent agreement among the non-faulty parties. The faulty behavior may range from simple mistakes to total breakdown to skillful adversarial talent. Attaining agreement on a common value is difficult as one does not know whom to trust. BA is used in almost any task that involves multiple parties, like voting, bidding, secure function evaluation, threshold key generation, MPC, etc.

The problem has drawn much attention over the years and many aspects of the problem have been studied considering various models [68, 18, 29, 39, 35, 118, 72, 110, 2, 24, 25, 26, 30, 31, 32, 44, 56, 54, 57, 59, 60, 61, 74, 70, 71, 65, 67, 78, 86, 89, 114, 117, 134, 136, 150, 148, 149].

1.1.3 Multiparty Computation (MPC)

MPC finds its root in the Millionaires's problem proposed by Yao [151] which is known as one of the classical two-party computation problems. The problem is like this: Two parties Alice and Bob want to know who is richer between the two. But neither of them wants to reveal his/her actual wealth to the other. This problem is easy to solve if an independent trusted third party is available. Both

Alice and Bob reveal their wealth to the third party, who can easily determine the richer between the two. Thus Alice and Bob can find out the richer between the two without actually knowing the other person's wealth. But unfortunately, the third party's service may not be available in real-life instances. Thus, in the absence of third party, a protocol executed between Alice and Bob can simulate the role of third party and this protocol is called as two-party computation protocol. Secure Multiparty Computation (MPC) is the generalization (first proposed in [95]) of two-party computation for n party settings.

MPC is a fundamental problem, both in distributed computing and cryptography. In a nutshell, the problem of MPC[1] can be stated as follows: There is a set of n parties (among which some are faulty/corrupted), who do not trust each other. Still these parties wish to compute some function of common interest of their local inputs, without revealing anything about their respective inputs except for what can be derived from the function output.

The problem of MPC is so relevant to practical cryptographic applications, almost any known cryptographic problem (e.g encryption, authentication, commitment, signatures, zero-knowledge, BA) can be viewed as a special case of this general problem. Thus MPC has the potential to serve as a general uniform paradigm for the study of cryptographic problems. Some of the real world special instances of MPC include:

1. *Electronic Voting*: Here the voters wish to jointly compute the sum of their votes, without revealing any individual vote;

2. *Privacy-preserving Statistics*: A set of companies wishes to compute statistics of some secret business data, without revealing individual data sets;

3. *Privacy-preserving Database Operations*: A database is distributed over several servers in such a way that any corrupted server has no information on the stored data of other good servers, but still the servers can jointly compute standard database operations, like union, intersection, finding cardinality etc.

MPC has been studied extensively in different settings (see [3, 19, 5, 6, 7, 20, 12, 13, 14, 21, 9, 36, 41, 48, 49, 52, 95, 93, 98, 101, 103, 104, 135, 138, 143] and their references).

1.2 Various Models for Studying VSS, BA and MPC

The problem of VSS, BA and MPC may assume many different forms, depending on the communication model (that talks about the attributes of the underlying network), the adversary model (that captures the nature, capacity and computing power of the adversary) etc.

[1]Note that this problem is sometimes called *secure function evaluation* (SFE) whereas the term multiparty computation would then refer to the more general problem of ongoing computations where several function evaluations might be intertwined. To be more clear, the kind of computation, in which all inputs can be given at the beginning of the computation is called SFE or non-reactive multiparty computation. On the other hand more general reactive multiparty computation allows to perform an arbitrary on-going (reactive) computation, where the users can give inputs and get outputs several times during the computation.

1.2.1 Communication Model

The prominent attributes of the underlying network that lead to the various classifications of VSS, BA and MPC are discussed below and are summarized in Table 1.1.

1.2.1.1 Medium of Communication (uni-cast channel or multi-cast channel or Broadcast channel)

In any protocol, the parties communicate with each other over channels where channels can be *uni-cast/point-to-point* (one to one), *multi-cast* (one to many) and *broadcast* (one to all). Uni-cast/point-to-point channel enables both way communication between two parties. Multi-cast channels allow a party to send some message identically to a subset of parties in the network. Broadcast channel allows any party to send some message identically to all other parties in the network. Uni-cast and broadcast channels are two extreme cases of multi-cast channels. We may consider the channels to be undirected and directed. Most of the literature on VSS, BA and MPC assume the existence of pairwise point-to-point channels among the parties and often, broadcast channels are also assumed. In many other cases, in the absence of broadcast channel, it is simulated by executing broadcast (a variant of BA problem) protocol. There are few works in the literature that considers multi-cast channels [45, 139]. *In general, when we say channel, we will usually mean uni-cast or point-to-point channel.*

1.2.1.2 Network Topology (Complete or Incomplete)

The topology of the network can be *complete* or *incomplete*. In a complete network, every pair of parties are directly connected, while in an incomplete network, the connectivity can be limited. Except a very few attempts [55, 138, 16, 17, 88], most of the works on VSS, BA and MPC consider complete network [20, 138, 43, 115].

1.2.1.3 Control over Channels (Secure or Insecure or Unauthenticated)

We distinguish three levels of control over the channels (or three levels of abstraction of the channel security): *Secure* (authentic and secret), *Insecure* (authentic but tappable), and *Unauthenticated* (unauthenticate and tappable). In *secure* channel model, the communication between any two uncorrupted or honest parties are completely out of reach to the adversary i.e adversary cannot affect or change or even eavesdrop the communication. Alternatively in *insecure* channel model, the adversary can hear all the communication among all the parties; yet the adversary can not alter the communication (between two honest parties). In the last alternative, called *unauthenticated* channel model, the adversary has full control over the communication. That is, on the top of tapping the communication the adversary can delete, generate and modify messages at wish. This parameter (i.e control over channel) can also be considered as the attribute of adversary rather than the attribute of network.

1.2.1.4 Synchrony of Network (Synchronous or Asynchronous or Hybrid)

Synchrony divides the networks into three types: *Synchronous*, *Asynchronous* and *Hybrid*. In a *synchronous* network, all the parties have access to a common global clock. All the messages are sent on a clock 'tick' and are received at the next clock 'tick'. That is, the delay of messages in the channel is bounded by a known constant. In *asynchronous* network, there is no global clock. Moreover, arbitrary (yet finite) time may lapse between the sending and receipt of a message. In particular the messages may be received in an order different than the order of sending. Thus in asynchronous network, the inherent difficulty in designing a protocol comes from the fact that when a party does not receive an expected message then he cannot decide whether the sender is corrupted (and did not send the message at all) or the message is just delayed in the network. So a party can not wait to consider the values sent by all parties before commencing its computation at any particular step, as waiting for all of them can turn out to be endless. Due to this, the protocols in asynchronous network are generally involved in nature and require new set of primitives. For an comprehensive introduction to asynchronous network and protocols, see [35].

There is another class of network called *hybrid* network that exercises the properties of synchronous and asynchronous network in many different ways. There are at least two different notions for hybrid network available in the literature: (a) A hybrid network allows a few synchronous rounds followed by a fully asynchronous communication [15]; (b) A hybrid network consists of a synchronization point and the network is asynchronous before and after the synchronization point [51]. The synchrony reflects some effects on the behavior of adversary as well. In *asynchronous* network, the adversary is given the power to schedule the delivery of *all* messages in the network. However, the adversary can *only schedule* the messages communicated between honest parties, without having any access to them (in secure channel model).

Table 1.1: The Attributes of Communication Model.

Medium of Communication	Network Topology	Control over Channels	Synchrony of Network
Uni-cast Channel *Multi-cast Channel* *Broadcast Channel*	*Complete* *Incomplete*	*Secure* *Insecure* *Unauthenticated*	*Synchronous* *Asynchronous* *Hybrid*

1.2.2 Adversary Model

Various models of VSS, BA and MPC can be obtained based on the kind of adversary. Some of the features which characterize the adversary are discussed below and are summarized in Table 1.2.

1.2.2.1 Computational Resources (Bounded or Unbounded)

The computational resources at the disposal of the adversary may be limited to *probabilistic polynomial time* as in *cryptographic* settings [95]. On the other hand adversary may have *unbounded computing power* as in *information theoretic* settings [20, 41]. In information theoretic settings, protocols can be either *perfectly secure or in short perfect* (error free) or *statistically secure or in short statistical* (involves negligible error probability).

1.2.2.2 Control over the Corrupted Parties (Passive or Fail-stop or Active or Mixed)

According to the type of control over the corrupted parties, adversary can be of four genres: *passive*, *fail-stop*, *active/Byzantine* and *mixed*. The adversary may act like an eavesdropper, that is he may gather all the information present with corrupted parties and perform any arbitrary computation on this gathered data in an effort to find out the honest party's data. Such an adversary is called as *passive* adversary. Furthermore if the adversary can stop the working of any of the corrupted parties, then he is referred to as a *fail-stop* adversary. In addition, if the adversary can also take complete control of the corrupted parties and alter the behavior of the corrupted parties in an arbitrary and coordinated fashion, he is called as *Byzantine or active* adversary. Lastly, an adversary may simultaneously control some parties in passive, fail-stop and active fashion (possibly disjoint set of parties); such a generalized adversary is called *mixed* adversary.

1.2.2.3 Mobility (Static or Adaptive/Dynamic or Mobile/Proactive)

Depending on the point in time when the adversary is allowed to corrupt parties, adversary can be of three types: *static*, *adaptive/dynamic* and *mobile/proactive*. If the adversary decides on the set of parties that it would corrupt before the protocol begins its execution, then such an adversary is referred to as a static adversary [41, 95]. Thus the set of corrupted parties is fixed (but typically unknown) during the whole computation. More generally, the adversary may be allowed to corrupt parties during the protocol execution, depending on the information gathered so far. Such an adversary is called adaptive or dynamic. Thus an adaptive or dynamic adversary [48] chooses which parties to corrupt as the computation proceeds. In both the above cases, once a party is corrupted, he remains corrupted for the rest of the protocol execution. Like an adaptive adversary, a mobile adversary can corrupt parties at any time, but he can also release corrupted parties, regaining the capability to corrupt further parties. Thus an adversary is mobile [124] if he can corrupt, in an adaptive way, a different set of parties at different times during the execution. That is a party once corrupted need not remain so throughout. Mobile adversaries model, for example, to virus attacks.

1.2.2.4 Corruption Capacity (Threshold or Non-threshold)

The number of parties that the adversary can keep corrupted at any given instance of time is his corruption capacity. There are two different ways of specifying the number of corrupted parties, viz. *threshold* and *non-threshold*. In the threshold specialization [19, 20, 95, 100], the number of corrupted parties, at any given time, is limited to at most t (a threshold). The non-threshold specialization is a generalization of the threshold one. In the non-threshold specialization [10, 50, 77, 2, 99, 100], an adversary structure which is a set of subsets of the parties, is used where the adversary is permitted to corrupt the parties of any one arbitrarily chosen subset in the adversary structure.

Table 1.2: The Attributes of Adversary Model.

Computational Resources	Control Over Corrupted Parties	Mobility	Corruption Capacity
Bounded (Cryptographic) *Unbounded (Information theoretic)*	*Passive* *Fail-stop* *Active/Byzantine* *Mixed*	*Static* *Adaptive* *Mobile*	*Threshold* *Non-threshold*

1.3 The Model of our Interest and Informal Definitions of the Problems

In this thesis, for VSS and MPC, we consider the following:

- **Communication Model**
  1. **Medium of Communication:** Point-to-point channel with and sometime without broadcast channel.
  2. **Network Topology:** Complete network
  3. **Control over Channels:** Secure channel model.
  4. **Synchrony of Network:** Both synchronous and asynchronous.
- **Adversary Model**
  1. **Computational Resources:** Unbounded powerful adversary, information theoretic security (both perfect and statistical)
  2. **Control over the Corrupted Parties:** Active/Byzantine
  3. **Mobility:** Static
  4. **Corruption Capacity:** Threshold

  We denote the adversary with the above features by $\mathcal{A}_t$, where t is the threshold for corruption.

For BA also we follow the same settings as above, except that we study it in only asynchronous network. Later we will discuss about our models more elaborately in individual chapters of this thesis and will present the formal definitions of the problems in respective model. For the time being, we just use the following informal description of the problems akin to our model.

- **VSS:** Informally, a VSS is a two phase protocol (Sharing and Reconstruction) carried out among n parties in the presence of adversary $\mathcal{A}_t$ who can maliciouslsly/actively corrupt up to t parties. The goal of the VSS protocol is to share a secret, s, among the n parties during the sharing phase in a way that would later allow for a unique reconstruction of this secret in the reconstruction phase, while preserving the secrecy of s from $\mathcal{A}_t$ until the reconstruction phase.
- **BA:** A protocol among a group of n parties (out of which t may be corrupted by $\mathcal{A}_t$), each having a private value, is said to achieve Byzantine agreement, if, at the end of the protocol, all honest parties agree on a value and the following conditions hold:

1. **Agreement:** All honest parties agree on the same value;
2. **Validity:** If all honest parties start with the *same* value $v \in \{0, 1\}$, then all honest parties agree on v;
3. **Termination:** All honest parties eventually agree.

- **MPC:** MPC allows a set of n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ to securely compute an agreed function f, even in the presence of centralized active adversary $\mathcal{A}_t$. More specifically, assume that f can be expressed as $f : \mathbb{F}^n \rightarrow \mathbb{F}^n$ and party P_i has input $x_i \in \mathbb{F}$, where $\mathbb{F}$ is a finite field. Now MPC ensures the following:

  1. **Correctness:** At the end of the computation of f, each honest P_i gets $y_i \in \mathbb{F}$, where $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, irrespective of the behavior of the corrupted parties.
  2. **Secrecy:** The adversary $\mathcal{A}_t$ should not get any information about the input and output of the honest parties, other than what can be inferred from the input and output of the corrupted parties.

  In any general MPC protocol, the function f is specified by an arithmetic circuit over $\mathbb{F}$, consisting of input, linear (e.g. addition), multiplication, random and output gates. We denote the number of gates of each type by c_I, c_A, c_M, c_R and c_O, respectively. *Among all the different types of gate, the evaluation/computation of a multiplication gate requires the most communication complexity. So the communication complexity of any general MPC is usually given in terms of the communication complexity per multiplication gate* [14, 13, 12, 52, 126].

1.4 History of Extant Literature on VSS, BA and MPC

We will present the history of each of the problems mostly restricting to the model of our interest. As far as the communication model is considered all the works that we quote here consider a complete network of n parties, pairwise connected by secure channels (sometimes broadcast channel is also assumed to be available; we will specify when it is so). For every problem, we divide the literature survey into two parts based on synchrony of the network; one part focusing on the works in synchronous network and other part concentrating on the works in asynchronous network. Before proceeding to the survey, it is important to know that in synchronous network any protocol has four system parameters or measures: *Resilience*, *Communication Complexity*, *Round Complexity* and *Computation Complexity.*

1. *Resilience:* It is the maximum number of corrupted parties that the protocol can tolerate and still satisfy its properties;

2. *Communication Complexity:* It is the total number of bits communicated by the honest parties in the protocol. A protocol is called communication efficient if the communication complexity is polynomial in n and error parameter (in case the protocol is statistical and has an error parameter).

3. *Round Complexity:* In synchronous network due to the existence of a global clock, the protocols operate in a sequence of rounds, where a round is defined as the time period between two consecutive 'tick's of the global clock. In each round, a party performs some local computation, sends new messages to the other parties through the private channels (and broadcasts some information over the broadcast channel), then it receives the messages that were sent by the other parties in this round on the private channels (and broadcast channels). Now round complexity is the total number of rounds taken for the execution of the protocol. A protocol is called round efficient if the round complexity is polynomial in n and the error parameter (in case the protocol is statistical and has an error parameter).

4. *Computation Complexity:* It is the computational resources required by the honest parties during a protocol execution. A protocol is called computationally efficient if the computational resources required by each honest party are polynomial in n and error parameter (in case the protocol is statistical and has an error parameter).

In asynchronous network there is no global clock and thus in general there is no concept of clock 'tick's or so called rounds. Here the time required for the execution of a protocol is quantified by the parameter called *Running Time*. Hence, apart from the parameters *resilience*, *communication complexity* and *computation complexity*, a protocol in asynchronous network has another parameter called *running time*.

- *Running Time*: We present an informal, but standard definition of the running time of an asynchronous protocol. For more detailed definition of running time, see [118]. The current definition is taken from [39, 35]. Consider a virtual 'global clock' measuring time in the network. Note that the parties cannot read this clock. Let the *delay* of a message be the time elapsed from its sending to its receipt. Let the *period* of a finite execution of a protocol be the longest delay of a message in the execution. The *duration* of a finite execution is the total time measured by the global clock divided by the period of the execution.

  Let E be an event that occurs in an execution of a protocol. Let *average duration* be the average over the random inputs of the parties, of the duration of executions of the protocol in which E occurs. Now the *expected running time* of a protocol, *conditioned on event E*, is the maximum over all inputs and applicable adversaries, of *average duration*s.

1.4.1 The History of VSS in Synchronous Network

As mentioned before VSS finds its root in secret sharing. Since the appearance of Shamir's [140] and Blackley's [27] seminal papers on threshold secret sharing, the research on this topic has been done extensively. The solutions of Shamir and Blackley worked in a model where there is no faults in the system. Tompa and Woll [147] and McEliece and Sarwate [121] gave the first partial solution considering faults in the model. Finally, Chor et al. [43] defined the complete notion of VSS and give the first ever solution for VSS. Since then, under various assumptions and driven by different motivations, solutions for VSS were proposed in [96, 64, 20, 41, 138, 133, 93, 90, 48, 91, 49, 73, 109]. While the works of

[96, 64, 133, 93] consider cryptographic model where the adversary has bounded computing power, the works of [20, 41, 138, 48, 91, 49, 73, 109] consider information theoretic model (i.e. under the assumption of a *computationally unbounded adversary*). The prominent works that consider generalized non-threshold adversary are [92, 90, 77]. Several other works that assumes a mobile adversary are [34, 97].

We now channelize our attention to the information theoretic model with threshold static adversary. In the literature restricted to this model, the basic approach of designing a VSS is to use Shamir's protocol as the backbone structure and then (on top of that) use some proof from the dealer that the values shared lie on a polynomial of degree t, thus ensuring that the shares identify a unique secret. In information theoretic settings, there are mainly two flavors of VSS: *perfect* VSS (i.e. error free) and *statistical* VSS (involves some probability of error).

- *Perfect VSS:* It is well known that perfect VSS tolerating $\mathcal{A}_t$ is possible iff $n \geq 3t + 1$ [55]. Perfect VSS protocols are proposed in [20, 91, 73, 109]. The protocol of [20] was designed with the motivation of using it in MPC protocol. The investigation on the exact round complexity of perfect VSS was first conducted by Gennaro et al. [91] and subsequently completed by [73, 109].

- *Statistical VSS:* On the other hand, statistical VSS tolerating $\mathcal{A}_t$ is achievable with $n \geq 2t + 1$ in the presence of a broadcast channel (in addition to point-to-point channel between every pair of parties) [138]. Statistical VSS protocols are proposed in [41, 138, 48, 49] in which, except [41], all other protocols were designed with optimal resilience i.e with $n = 2t + 1$ parties. The protocol of [138, 48, 49] was designed with the motivation of using them in MPC protocol. The works of [48, 49] strive for designing statistical VSS protocols with better communication complexity (not bothering too much on their round complexity). So far the round complexity of statistical VSS is not investigated yet.

1.4.2 The History of VSS in Asynchronous Network

In comparison to the VSS protocols in synchronous network, research in VSS in asynchronous network has received much less attention. In information theoretic settings, there are two flavors of asynchronous VSS (now onwards we call it as AVSS): *perfect* AVSS (i.e error free) and *statistical* AVSS (involves negligible error probability).

- *Perfect AVSS:* Perfect AVSS tolerating $\mathcal{A}_t$ is possible if and only if $n \geq 4t + 1$ [19, 35]. Hence, we call any *perfect* AVSS protocol with $n = 4t + 1$ as *optimally resilient, perfect* AVSS protocol. Such AVSS protocols are proposed in [19, 35, 13]. The protocols of [19, 35, 13] were proposed to design MPC protocol in asynchronous network.

- *Statistical AVSS:* Statistical AVSS tolerating $\mathcal{A}_t$ is possible if and only if $n \geq 3t + 1$ [39, 21]. To the best of our knowledge, the AVSS protocol of [39, 21] is the only known *optimally resilient statistical* AVSS protocol (i.e., with $n = 3t + 1$). There is one AVSS protocol with non-optimal resilience (with $n = 4t+1$ parties) reported in [66, 67]. The AVSS protocol of [39, 66] were designed for building asynchronous BA with $n = 4t + 1$ parties.

1.4.3 The History of MPC in Synchronous Network

The MPC problem was first introduced by Yao [151] in two party settings. The first generic solutions presented in [95, 42, 85] in n party settings with $n > 2$, were based on cryptographic intractability assumptions. Later MPC with information-theoretic security in n party settings were presented in [20, 41, 138, 4, 6]. Much like VSS, in information theoretic settings, there are two flavors of MPC: *perfect* MPC (i.e. error free) and *statistical* MPC (involves some probability of error).

- *Perfect MPC:* Perfect MPC protocol tolerating $\mathcal{A}_t$ is possible if and only if $n \geq 3t + 1$ [20]. Perfect MPC with optimal resilience (i.e with $n = 3t + 1$ parties) has been studied in [20, 3, 7, 5, 98, 111, 14]. While works of [3, 7] were focused to design MPC with constant round complexity at the cost of very high communication complexity, works of [98, 14] focus on improving communication complexity at the expense of high round complexity. The protocols of [98, 14] uses player elimination framework proposed by Hirt et al. in [98].
- *Statistical MPC:* Statistical MPC tolerating $\mathcal{A}_t$ is possible when $n \geq 2t + 1$ [138, 4, 6], provided that a common broadcast channel is available (in addition to point-to-point channel between every pair of parties). Statistical MPC designed with exactly $n = 2t+1$ parties (in the presence of a broadcast channel, along with secure point-to-point channel between every two parties) is said to have optimal resilience. Statistical MPC protocols with optimal resilience are reported in [138, 4, 3, 6, 48, 49, 12]. There are several works reported on statistical MPC with non-optimal resilience (i.e with $n > 2t + 1$ parties) in [41, 101, 52] (these protocols were designed with $n \geq 3t + 1$ parties). The protocols of [101, 52] use player elimination framework. The protocol of [12] uses *dispute control* framework which is actually a generalization of *player elimination*.

There are several attempts in the literature when specific instances of MPC problems are studied such as — multiparty set intersection, set union, set cardinality, other set related problems [84, 116], Longest Common Subsequence (LCS) problem [81], private stable matching [80], secure group barter [82] etc. Though these problems can be solved using general MPC protocol; but the disadvantages are that a general MPC may not give as efficient solution as a specific solution to these problems may provide. This is because, the specific solution takes into account the nuances and subtleties of the problem and accordingly finds efficient solution.

1.4.4 The History of MPC in Asynchronous Network

Unlike MPC in synchronous network, designing asynchronous MPC (now onwards we call it as AMPC) protocols has received less attention due to their inherent difficulty. In information theoretic settings, AMPC protocols can be categorized mainly into two types:

- *Perfect AMPC:* In [19], it is shown that perfect AMPC tolerating $\mathcal{A}_t$ is possible iff $n \geq 4t+1$. Thus any perfect AMPC designed with $n = 4t+1$ is said to be *optimally resilient*. Optimally resilient, perfect AMPC protocols are reported in [19, 143, 13].

- *Statistical AMPC:* From [21], it is known that statistical AMPC tolerating $\mathcal{A}_t$ is possible iff $n \geq 3t+1$. Thus any statistical AMPC protocol designed with $n = 3t+1$ is said to be *optimally resilient*. Optimally resilient, statistical AMPC is reported in only [21]. In comparison to perfect AMPC, statistical AMPC protocol [21] in the literature has much more communication complexity. To achieve better communication complexity for the statistical AMPC protocols, researchers have tried to design statistical AMPC with non-optimal resilience i.e with $n = 4t+1$ parties. Such AMPC protocols are reported in [135] and recently in [107]. Both the AMPC protocols of [135] and [107] are based on *player elimination* framework of [98], an important technique introduced in synchronous network in order to reduce communication complexity of MPC protocols.

Recently in [15], the authors have designed communication efficient MPC protocols over networks that exhibit partial asynchrony (where the network is synchronous up to certain point and becomes completely asynchronous after that). In another work, Damgård et al. [51] have reported efficient MPC protocol over a network that assumes the concept of synchronization point; i.e. the network is asynchronous before and after the synchronization point.

1.4.5 The History of BA in Asynchronous Network

In information theoretic settings, any asynchronous BA (now onwards we call it as ABA) protocol tolerating $\mathcal{A}_t$ is possible iff $n \geq 3t+1$ [132, 118]. Thus any ABA protocol designed with $n = 3t+1$ parties is called as *optimally resilient*. By the seminal result of [71], any ABA protocol, irrespective of the value of n, must have some *non-terminating* runs/executions, where some honest party(ies) may not output any value and thus may not terminate at all. So we say an ABA protocol to be $(1-\epsilon)$-terminating, if the *probability* of occurrence of non-terminating executions in the protocol is $(1-\epsilon)$. On the other hand, we call an ABA protocol to be *almost-surely terminating*, a term coined by Abraham et al. in [1], if the *probability* of occurrence of a non-terminating execution in the protocol is *asymptotically zero*.

Rabin [136] and Ben-Or [18] presented ABA protocols with $n \geq 8t+1$ and $n \geq 5t+1$ respectively. Since, both these protocols were not *optimally resilient*, researchers have tried to design ABA protocol with *optimal resilience* or close to optimal resilience. The ABA protocols of [29, 39, 1] are designed with optimal resilience, whereas the protocol of [66] is designed with $n = 4t+1$ parties. The protocol of [29] requires exponential ($\Theta(2^n)$) expected time and exponential ($\Theta(2^n)$) communication complexity. But, the protocols of [66, 39, 1] require polynomial communication complexity. The protocols of [29, 1, 66] are *almost-surely terminating* and the protocol of [39] is $(1-\epsilon)$-terminating. Finally, protocols of [39, 66] require constant expected running time and the protocol of [1] requires polynomial ($\mathcal{O}(n^2)$) expected running time. All the above protocols have very high communication complexity and thus designing communication efficient and communication optimal ABA is still a challenging task to achieve.

1.5 The Contribution of this Thesis

This section is devoted for the description of our contributions for VSS, BA and MPC. Prior to the details of our contribution, we present the following important discussion. Many of our protocols for VSS, BA and MPC are statistical in nature which means they involve negligible error probability of ϵ in their computation. *By negligible it means that ϵ is exponentially small in n i.e $\epsilon \leq \frac{1}{2^n}$ or $\epsilon \leq \frac{1}{2^{\alpha n}}$ for some integer α greater than or equal to one.* Now to bound the error probability of a protocol by some desired value of ϵ, we have to choose an appropriate finite field over which all the computations of the protocol should be carried out. We say that the protocol should operate on finite Galois field $\mathbb{F} = GF(2^\kappa)$ where the κ has to be chosen based on the desired value for ϵ and the relation between ϵ and κ. Here κ is called as the error parameter. Due to the different working of protocols, there will be different relationships between κ and ϵ for different protocols (therefore we mention them in appropriate context in respective chapters). From $\epsilon \leq \frac{1}{2^{\alpha n}}$, we may conclude that $n = \mathcal{O}(\log \frac{1}{\epsilon})$. This relation will be used throughout this thesis.

Now the following are our contributions for VSS, MPC and BA:

1.5.1 Investigation on The Round Complexity of Statistical VSS

Round complexity is one of the most important complexity measures of interactive protocols. In this thesis, we study the round complexity of statistical VSS and show that existing lower bounds for perfect VSS can be circumvented by allowing negligible error probability in the protocol executions.

The study of the round complexity of VSS in the information theoretic setting, was initiated by Gennaro et al. [91]. Their investigation was conducted for *perfect VSS* i.e under the assumption that the protocols are *error-free*. The assumed network model was a complete synchronous network with pairwise secure channels and a broadcast channel. They refer to the round complexity of VSS as the number of rounds in the sharing phase and prove that

1. A 1-round sharing VSS is possible if and only if $t = 1$ and $n \geq 5$.
2. A 2-round sharing VSS is possible if and only if $n \geq 4t + 1$.
3. A 3-round sharing VSS is possible if and only if $n \geq 3t + 1$.

In this thesis, we examine the round complexity of statistical VSS and also investigate the question of whether the lower bounds for the round complexity of perfect VSS can be overcome by introducing a negligible probability of error. We answer this in affirmative by showing that

1. A 1-round sharing VSS is possible if and only if $t = 1$ and $n \geq 4$.
2. A 2-round sharing VSS is possible if and only if $n \geq 3t + 1$.

Our results clearly show that probabilistically relaxing the conditions of VSS helps to increase the fault tolerance. Our protocols have two rounds in the reconstruction phase and interestingly two rounds in reconstruction phase can be collapsed into a single round when the adversary is considered to be non-rushing[2].

[2]A rushing adversary can wait to hear the incoming messages in a given round prior to sending out its own messages

A weaker version of VSS is called WSS (Weak Secret Sharing) which is generally used as a tool to design VSS protocols [138, 137]. The study of the round complexity of perfect WSS in the information theoretic setting, was initiated in [73]. In [73], the authors have referred to the round complexity of WSS as the number of rounds in the sharing phase and have shown that

1. Efficient 1-round as well as 2-round sharing WSS protocol is possible if and only if $n \geq 4t + 1$.

2. Efficient 3-round sharing WSS protocol is possible if and only if $n \geq 3t + 1$.

In this thesis, we completely resolve the round complexity of WSS involving negligible error probability by showing that

1. Efficient 1-round as well as 2-round sharing WSS protocol is possible iff $n \geq 3t + 1$.

2. Efficient 3-round sharing WSS protocol is possible iff $n \geq 2t + 1$.

Our results clearly show that probabilistically relaxing the conditions of WSS helps to increase the fault tolerance.

A part of the above results has appeared in [125]. The lower bound proof for 2-round sharing WSS presented in this thesis, is based on [112].

1.5.2 Study of Communication and Round Efficiency of Statistical VSS

In addition to round complexity, communication complexity is another important parameter of VSS protocols. We study the communication and round efficiency of statistical VSS protocols with optimal resilience (that is with $n = 2t+1$ provided a public broadcast channel is available). We propose an optimally resilient, statistical VSS scheme that is better than all the existing optimally resilient, statistical VSS protocols both in terms of communication as well as round complexity.

There are three optimally resilient statistical VSS schemes reported so far, namely the schemes of [138], [48] and [49]. Among the three protocols, the protocols of [48] and [49] provide the best known communication complexity. Both of them share a single secret with a communication complexity of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits (communication over both pairwise secure and broadcast channel), where ϵ is the error probability of the protocol.

In this thesis, we propose a protocol that provides a communication complexity of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits (communication over both pairwise secure and broadcast channel) for sharing ℓ secrets simultaneously. So for large value of ℓ, our protocol is better than the protocols of [48] and [49]. Additionally, our protocol provides better round complexity than all the three existing protocols.

The key tool that is used for constructing our VSS is an efficient Information Checking Protocol (ICP). ICP is a tool for authenticating messages in the presence of computationally unbounded corrupted parties. We present an ICP that provides the best known round as well as communication complexity so far in the literature.

1.5.3 Study of Communication Efficiency of Statistical AVSS

We study the communication efficiency of statistical AVSS protocols. In this thesis, we design three statistical AVSS protocols: (a) two with optimal resilience i.e with $n = 3t+1$; (b) one with non-optimal resilience i.e with $n = 4t+1$. Our protocols are highly communication efficient and show significant improvement over existing statistical AVSS protocols.

1. **Communication Efficient AVSS with Optimal Resilience:** Between our two protocols with *optimal resilience*, the first one communicates $\mathcal{O}((\ell n^3 + n^4 \log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits to simultaneously share $\ell \geq 1$ secrets, where ϵ is the error probability of the protocol. The other protocol communicates $\mathcal{O}((\ell n^3 + n^4)\log\frac{1}{\epsilon})$ bits to simultaneously share $\ell \geq 1$ secrets. The second protocol has a limitation as well as advantage compared to the first one. The limitation is that a corrupted dealer may commit a NULL value (we will discuss about this in more detail later in this thesis) and get away with this, whereas in the first protocol dealer is forced to commit some secret from the working field. The advantage is that the second protocol is much simpler than the first one. The second protocol is used in our ABA and we show that it's NULL commitment is enough for ABA protocol. The first protocol is suitable for AMPC and we use it for designing our AMPC protocol. There is only one known statistical AVSS protocol with $n = 3t+1$ reported in [39]. The AVSS protocol of [39] requires a communication complexity of $\mathcal{O}(n^9(\log\frac{1}{\epsilon})^4)$ bits to share a *single* secret. Thus our AVSS protocols show significant improvement in communication complexity over the AVSS protocol of [39].

2. **Communication Efficient AVSS with Non-optimal Resilience:** Our statistical AVSS with $n = 4t+1$ achieves $\mathcal{O}((\ell n^2 + n^3)\log n)$ bits of communication complexity for sharing ℓ secrets concurrently.

Our statistical AVSS protocols with optimal resilience appeared in [128, 127].

1.5.4 Study of Communication Efficiency of Perfect AVSS

We also study the communication efficiency of perfect AVSS protocols. In this thesis, we design a perfect AVSS with optimal resilience i.e with $n = 4t+1$, that provides the best communication complexity in the literature of perfect AVSS.

Our perfect AVSS protocol achieves an amortized communication cost $\mathcal{O}(n \log n)$ bits for sharing a single secret. So far the best known AVSS with $4t+1$ was proposed by [13]. The protocol of [13] is perfect in nature and requires an amortized communication cost $\mathcal{O}(n^2 \log n)$ bits for sharing a single secret.

Our perfect AVSS protocol is presented in [131].

1.5.5 Study of Communication and Round Efficiency of Statistical MPC

The round and communication complexity are the most important complexity measures of MPC protocols in synchronous network. A proper balance of both the complexity measures is essential from the perspective of practical implementation of MPC protocol. So far communication complexity wise the best known optimally resilient statistical MPC is reported in [12]. The protocol of [12] achieves

$\mathcal{O}(n^2 \log \frac{1}{\epsilon})$ bits of private communication [7] per multiplication gate at the cost of high round complexity of $\mathcal{O}(n^2\mathcal{D})$, where $\mathcal{D}$ is the multiplicative depth of the arithmetic circuit representing function f and ϵ is the error probability of the protocol. On the other hand, round complexity wise best known optimally resilient statistical MPC protocols are presented in [4, 5] and [138][8]. The protocols of [4, 5] and [138] have round complexity of $\mathcal{O}(\mathcal{D})$. But unfortunately, these MPC protocols require broadcasting[9] of $\Omega(n^5(\log \frac{1}{\epsilon})^4)$ bits per multiplication gate[10].

In this thesis, we focus to balance both the complexity measures of statistical MPC. With this aim in mind, we present a new optimally resilient statistical MPC that acquires a round complexity of $\mathcal{O}(\mathcal{D})$ and broadcasts $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits per multiplication gate. Hence our protocol maintains the round complexity of most round efficient protocol while improving the communication complexity. Moreover, for all functions with constant multiplicative depth, our protocol achieves constant round complexity while most communication efficient MPC of [12] requires $\mathcal{O}(n^2)$ rounds.

The above results are based on [126, 130].

1.5.6 Designing Efficient Multiparty Set Intersection Protocol in Synchronous Network

In information theoretic settings, a protocol for multiparty set intersection (MPSI) allows a set of n parties, each having a set of size m to compute the intersection of those sets, even in the presence of $\mathcal{A}_t$. In this thesis, we re-visit the problem of MPSI in information theoretic settings. In [116], Li et al. proposed a statistical MPSI protocol with $n = 3t + 1$ parties and claimed that their protocol takes six rounds of communication and communicates $\mathcal{O}(n^4m^2)$ field elements. However, we show that the round and communication complexity of the protocol in [116] is much more than what is claimed in [116].

We then propose a *novel* statistical MPSI protocol with $n = 3t + 1$ parties, which significantly improves the "actual" round and communication complexity of the protocol given in [116]. To design our protocol, we use several tools including a VSS protocol, which are of independent interest. But the protocol of [116] and our proposed protocol with $n = 3t + 1$ are statistical and still they require $n = 3t + 1$ parties. Thus the protocols are non-optimal in resilience. So we also design an optimally resilient statistical MPSI protocol with $n = 2t + 1$.

A major part of the above results has appeared in [129, 130].

1.5.7 Study of Communication Efficiency of Statistical AMPC

In this thesis, we work on the communication efficiency of statistical AMPC protocols and design two protocols, one with optimal resilience i.e $n = 3t + 1$ and the other one with non-optimal resilience (with $n = 4t+1$). Our statistical AMPC with optimal resilience shows huge improvement in communication complexity over the only known statistical AMPC with optimal resilience reported in [21].

---

[7] Communication over secure channels.

[8] We have considered MPC protocols with polynomial (in n and $\log \frac{1}{\epsilon}$) communication complexity. Constant round MPC can be achieved following the approach of [3] but at the expense of exponential blow-up in communication complexity.

[9] Communication over broadcast channel

[10] The authors of [49] claimed to have an optimally resilient statistical MPC protocol with round complexity of $\mathcal{O}(\mathcal{D})$ and communication complexity of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits of broadcast per multiplication gate, without providing exact implementation details.

1. **Communication Efficient Statistical AMPC with Optimal Resilience:** Our *statistical* AMPC protocol with *optimal resilience* communicates $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits *per multiplication gate*, where ϵ is the error probability of the protocol. The only known optimally resilient statistical AMPC of [21] communicates $\Omega(n^{11}(\log \frac{1}{\epsilon})^4)$ bits per multiplication gate. For designing our AMPC, we propose a new primitive called Asynchronous Complete Secret Sharing (ACSS). The ACSS protocol uses our statistical AVSS with $n = 3t + 1$ parties as an important building block.

2. **Communication Efficient Statistical AMPC with Non-optimal Resilience:** Communication complexity, being one of the important parameters of AMPC protocol, drew quite a bit of attention and hence there are a number of attempts to improve the communication complexity of AMPC protocols with $4t + 1$ parties. The latest such attempt is reported in [107] where the authors presented a statistical AMPC protocol with $n = 4t + 1$ that communicates $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits per *multiplication gate*, where $\mathbb{F}$ is the field over which computation is carried out. However, in this thesis we show that the protocol of [107] is not a correct statistical AMPC. We then present a new, simple, statistical AMPC protocol with $n = 4t + 1$ which communicates $\mathcal{O}(n^2 \log n)$ bits per *multiplication gate*.

Our *statistical* AMPC protocol with *optimal resilience* has appeared in [128].

1.5.8 Study of Communication Efficiency of Perfect AMPC

In this thesis, we also present a perfect AMPC protocol with optimal resilience that attains the best known communication complexity among all AMPC protocols designed with $n = 4t + 1$. Our protocol communicates $\mathcal{O}(n^2 \log n)$ bits per *multiplication gate*. We note that our perfect AMPC protocol is able to achieve the same communication complexity as our statistical AMPC protocol with $n = 4t + 1$; moreover it is now optimally resilient (that is, it is designed with $n = 4t+1$ parties) where our statistical AMPC protocol was non-optimal in resilience. The best known perfect AMPC protocol with optimal resilience [13] communicates $\mathcal{O}(n^3 \log n)$ bits per multiplication gate. Hence our AMPC protocol provides the best communication complexity among all the known AMPC protocols. For designing our perfect AMPC protocol we use our perfect AVSS with $n = 4t + 1$ parties.

This result has appeared in [131].

1.5.9 Designing Communication Efficient ABA for Small Message

An important variant of BA is Asynchronous Byzantine Agreement (ABA). The communication complexity of ABA is one of its most important complexity measures. In this thesis, we study the communication efficiency of ABA protocol for both the cases, namely ABA with optimal resilience i.e with $n = 3t + 1$ parties and ABA with non-optimal resilience i.e with $n = 4t + 1$ parties.

1. **Communication Efficient ABA with Optimal Resilience:** In this thesis, we present a simple and efficient ABA protocol whose communication complexity is significantly better than the communication complexity of the existing ABA protocols in the literature. Our protocol is *optimally resilient*

and thus requires $n = 3t+1$ parties for its execution. Moreover, our protocol is $(1-\epsilon)$-terminating.

Specifically, the *amortized* communication complexity of our ABA is $\mathcal{O}(\mathcal{C}n^4 \log \frac{1}{\epsilon})$ bits for attaining agreement on a *single* bit, where ϵ denotes the error probability of non-termination and $\mathcal{C}$ denotes the *expected running time* of our protocol. Conditioned on the event that our ABA protocol terminates, it does so in constant expected time; i.e., $\mathcal{C} = \mathcal{O}(1)$. Comparing our result with most recent optimally resilient, ABA protocols proposed in [39] and [1], we see that our protocol gains (in terms of communication complexity) by a factor of $\mathcal{O}(n^7(\log \frac{1}{\epsilon})^3)$ over the ABA of [39] and by a factor of $\mathcal{O}(n^4 \frac{\log n}{\log \frac{1}{\epsilon}})$ over the ABA of [1].

For designing our efficient ABA protocol, we use one of our statistical AVSS protocol with $n = 3t+1$. Our AVSS shares multiple secrets *concurrently* and is far better than multiple parallel executions of AVSS sharing single secret. Thus our AVSS brings forth several advantages of *concurrently* sharing multiple secrets.

The common coin primitive is one of the most important building blocks for the construction of ABA protocol. The only known efficient (i.e polynomial communication complexity) common coin protocol [67, 35] uses AVSS sharing a single secret as a black-box. Unfortunately, this common coin protocol does not achieve its goal when multiple invocations of AVSS sharing single secret are replaced by single invocation of our AVSS sharing multiple secrets. Hence in this thesis, we twist the existing common coin protocol to make it compatible with our new AVSS. As a byproduct, our new common coin protocol is much more communication efficient than the existing common coin protocol.

2. **Communication Efficient ABA with Non-optimal Resilience:** We have also studied the communication complexity of ABA with $n = 4t+1$ parties i.e with non-optimal resilience. We present an efficient ABA protocol with $n = 4t+1$ whose communication complexity is significantly better than the communication complexity of the only known existing ABA protocol of [66, 67] with $n = 4t+1$. Specifically, our ABA achieves an amortized communication complexity of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits for attaining agreement on a single bit, where $\mathbb{F}$ with $|\mathbb{F}| \geq n$ denotes the finite field over which our protocol performs all the computations. On the other hand, the only known ABA with $4t+1$ proposed in [66, 67] communicates $\Omega(n^4 \kappa \log |\mathbb{F}|)$ bits for single bit message, where κ is the error parameter. Like the ABA of [66, 67], our protocol has constant expected running time and also our protocol is *almost-surely terminating*. We use our perfect AVSS with $n = 4t+1$ as a vital building block for designing our ABA protocol.

Our result on ABA with optimal resilience has been published in [127].

1.5.10 Designing Communication Optimal ABA for Long Message

A-cast is the parallel notion of broadcast in asynchronous network. It allows a party to send some information identically to all other parties in the network. Though all existing protocols for A-cast and ABA are designed for a single bit message, in real life applications typically A-cast and ABA are invoked on *long* *message* (whose size can be in gigabytes) rather than on single bit. Therefore, it is important to design efficient *multi-valued* A-cast and ABA protocols (i.e protocols with *long message*) which extract several advantages offered by directly dealing with long messages and are far better than multiple invocations to existing protocols for single bit [72, 75]. In this thesis, we design new and highly efficient multi-valued A-cast and ABA protocols for long messages, based on access to the existing A-cast and ABA protocols for short messages. Moreover, we show that both our A-cast and ABA protocols are *communication optimal*, optimally resilient and are strictly better than existing protocols in terms of communication complexity for sufficiently large ℓ. In brief, we present the following results:

1. For an error parameter κ, we design a new, optimally resilient, multi-valued A-cast protocol with $n = 3t + 1$ that requires a private communication of $\mathcal{O}(\ell n)$ bits for an ℓ bit message, where ℓ is sufficiently large. Our A-cast protocol uses the existing A-cast protocol of [29] as a black box for smaller size message. The protocol of [29] is the only known protocol for A-cast and it requires a private communication of $\mathcal{O}(n^2)$ bits for a single bit message where $n = 3t + 1$.

2. For an error parameter κ, we design a new, optimally resilient, multi-valued ABA protocol with $n = 3t + 1$, which requires a private communication of $\mathcal{O}(\ell n)$ bits to agree on an ℓ bit message, where ℓ is sufficiently large. Our protocol uses the best known communication efficient ABA protocol presented in this thesis as a black box, which requires a *private* communication of $\mathcal{O}(n^7\kappa)$ bits to agree on a $(t + 1)$ bit message.

Our protocols are based on several new ideas. Fitzi et al. [75] are the first to design *communication optimal* multi-valued Byzantine Agreement (BA) protocols for large message with the help of BA protocols for smaller message, in *synchronous network*. Achieving the same in asynchronous network was left as an interesting open question in [75]. Our results in this thesis mark a significant progress on the open problem by giving protocols with a communication complexity of $\mathcal{O}(\ell n)$ bits for large ℓ. Moreover, to the best of our knowledge, ours is the first ever attempt to design multi-valued A-cast and ABA protocols, using existing A-cast and ABA protocols (for small messages) as a black-box.

1.6 The Organization of this Thesis

We divide the thesis in three parts. The first part consisting of five chapters i.e Chapter 2 - 6, includes all our results in synchronous network. The second part consisting of eight chapters i.e Chapter 7 - 14, takes care of all our results in asynchronous network. The third part consisting of Chapter 15 concludes this thesis with the summary of our results and future directions for pursuing research in VSS, BA and MPC. In the following, we brief the chapter wise contents.

In Chapter 2, we present a very important tool called Information Checking Protocol (ICP) which has been witnessed to play an important role in constructing VSS and WSS protocols. Our ICP will also be used as a building block to design several of our VSS and WSS protocols proposed in the next four chapters, namely Chapter 3, 4, 5, and 6. Our ICP provides the best known round and communication complexity so far in the literature.

In Chapter 3, we study the round complexity of statistical VSS and WSS protocols. Apart from presenting our lower bound results, we also present a set of new and novel statistical VSS and WSS protocols some of which use ICP of Chapter 2 as a black box.

In Chapter 4, we concentrate on designing statistical VSS protocol that is simultaneously communication efficient as well as round efficient. In the previous chapter, we were concerned on the round complexity of statistical VSS and WSS protocols and therefore communication complexity was of low priority than round complexity. Due to this, the protocols presented in the previous chapter were not designed keeping the communication efficiency in mind. In this chapter, we give importance to both the complexity measures simultaneously. Specifically, here we design statistical VSS with optimal resilience i.e with $n = 2t+1$ parties (plus a broadcast channel is available) that achieves the best known communication and round complexity in the literature. Our VSS uses the ICP presented in Chapter 2 as a vital black box primitive.

In Chapter 5, we present a new optimally resilient, statistical MPC protocol that simultaneously minimizes the communication and round complexity. The key tool for our new MPC is the statistical VSS protocol presented in Chapter 4. Using our VSS protocol, we propose a new and robust multiplication protocol for generating multiplication triples.

In Chapter 6, we re-visit the problem of MPSI in information theoretic settings. We first show that the actual round complexity and communication complexity of the statistical MPSI protocol proposed in [116] is much more than what is claimed in the paper [116]. We then propose a *novel* statistical MPSI protocol with $n = 3t + 1$ parties, which significantly improves the "actual" round and communication complexity of the protocol given in [116]. To design our protocol, we use several tools including a VSS protocol, which are of independent interest. Finally, we design an optimally resilient (i.e with $n = 2t + 1$) statistical MPSI protocol, borrowing the techniques from our proposed statistical MPC with $n = 2t + 1$ parties, presented in Chapter 5.

In Chapter 7, we present two novel asynchronous ICP abbreviated as AICP. Similar to the case of ICPs in synchronous network, our AICPs are used as vital tools for designing AVSS protocols in Chapter 8 which are further used in our ABA protocol (presented in Chapter 9) and statistical AMPC protocol (presented in Chapter 10).

In Chapter 8, we design two novel statistical AVSS protocols with optimal resilience (i.e with $n = 3t + 1$) using AICPs as black box. Both our AVSS protocols can share multiple secrets simultaneously (when necessary) and thus achieve many advantages offered by sharing multiple secrets concurrently in a single shot. Our AVSS protocols are far better than the existing protocols with $3t+1$ in terms of communication complexity. The protocols achieve different properties according to which we use one of the AVSSs in our ABA protocol presented in Chapter 9 and the other AVSS in our statistical AMPC protocol presented in Chapter 10.

In Chapter 9, we design an optimally resilient (i.e with $n = 3t + 1$ parties), communication efficient ABA protocol for small messages. As a key tool, we use one of our communication efficient AVSS (from Chapter 8). Furthermore, we also propose a new common-coin protocol using our AVSS. Our ABA reports the best known communication complexity among the existing ABA protocols with $n = 3t + 1$ parties.

In Chapter 10, we design an optimally resilient, communication efficient statistical AMPC protocol. First using one of our statistical AVSS of Chapter 8, we introduce a new asynchronous primitive called ACSS (Asynchronous Complete Secret Sharing) and design a protocol for it. Then using ACSS, we design our AMPC protocol. Our statistical AMPC protocol is the best among the existing statistical AMPC protocols with optimal resilience in terms of communication complexity.

In Chapter 11, we present two AVSS protocols: (a) one protocol is statistical and non-optimal in resilience (i.e designed with $n = 4t+1$ parties) (b) the other AVSS protocol is perfect and is designed with optimal resilience i.e $n = 4t + 1$ parties. Both the protocols are highly communication efficient and achieve some properties which are never attained by any AVSS protocols in the literature. Moreover the amortized communication complexity of the protocols for sharing a single secret is best in the history of AVSS with $4t + 1$ parties. These protocols are used in Chapter 12 for designing AMPC protocols with $n = 4t+1$ parties.

In Chapter 12, our focus is on AMPC protocols designed with $4t+1$ parties. The most communication efficient AMPC was reported in [107]. The AMPC is statistical in nature and claims to achieve a communication complexity of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits per *multiplication gate*, where $\mathbb{F}$ is the finite field over which all the computations of the protocol are carried out. In this chapter, we first show that the protocol of [107] is not a correct statistical AMPC. We then present a new, simple, statistical AMPC protocol with $n = 4t + 1$ which communicates $\mathcal{O}(n^2 \log n)$ bits per *multiplication gate*. Moving a step forward, we also present a perfectly secure AMPC protocol which communicates $\mathcal{O}(n^2 \log n)$ bits per *multiplication gate*. Now it is important to note that our perfect AMPC protocol is optimally resilient and at the same time achieves the same communication complexity as our statistical AMPC protocol. To design our AMPC protocols, we use our AVSS protocols presented in Chapter 11.

In Chapter 13, we design an efficient ABA protocol with $4t + 1$ parties. Our protocol shows significant improvement in communication complexity over the only known ABA protocol of [66, 67] with $n = 4t + 1$, while keeping all other properties in place such as constant running time and almost-surely terminating property. In fact our ABA in this chapter uses our perfect AVSS protocol presented in Chapter 11 as an important building block. Thus our ABA shows another application of the perfect AVSS protocol of Chapter 11 apart from the perfect AMPC protocol presented in Chapter 12.

In Chapter 14, we design an optimally resilient, communication optimal A-cast and ABA protocol for sufficiently long messages. Specifically, for an error parameter κ, our A-cast and ABA protocols with $n = 3t + 1$ require a private communication of $\mathcal{O}(\ell n)$ bits for ℓ bit message, where ℓ is sufficiently large. Our protocol uses player elimination framework introduced in [98] in the context of MPC. So far player-elimination was used only in MPC and AMPC and hence our result shows the first non-MPC application of the technique. Apart from this, we propose a novel idea to expand a set of $t+1$ parties, with all the honest party(ies) in it holding a common message m, to a set of $2t + 1$ parties with all honest parties in it holding m. Moreover, the expansion process requires a communication complexity of $\mathcal{O}(\ell n + poly(n, \kappa))$ bits, where $|m| = \ell$. We hope that this technique may be useful in designing protocol for many other form of consensus/Byzantine Agreement problems in asynchronous network that aims to achieve good communication complexity.

In Chapter 15, we present the summary of our results and future directions for pursuing research in VSS, BA and MPC.

Part I

Results in Synchronous Network

Chapter 2

An Efficient Information Checking Protocol

In this chapter, we present a very important primitive called Information Checking Protocol (ICP) which plays an important role in constructing statistical VSS and WSS protocols. Our ICP will also be used as a building block to design several of our statistical VSS and WSS protocols proposed in the next four chapters. Informally, ICP is a tool for authenticating messages in the presence of *computationally unbounded* corrupted parties. In this chapter, we focus on ICP in synchronous network and later in Chapter 7, we will focus on ICP in asynchronous network and present a couple of protocols for it. Here we extend the basic bare-bone definition of ICP, introduced by Rabin et al. [138] and then present an ICP that attains the best communication complexity and round complexity among all the existing ICPs in the literature. We also show that our ICP satisfies several interesting properties such as linearity property which is an important requirement in many applications of ICP as will be demonstrated later in this thesis.

2.1 Introduction

2.1.1 Existing Literature and Existing Definition of ICP

The notion of ICP was first introduced by Rabin et al. [138]. Rabin et al. [138] have used ICP for constructing a statistical WSS protocol which was further used to design a statistical VSS protocol with $n = 2t + 1$. Since then many ICPs have been designed [138, 39, 48] and used in constructing various statistical VSS and WSS protocols.

As described in [138, 39, 48], an ICP is executed among three parties: a *dealer* D, an *intermediary* INT and a *verifier* R. The dealer D hands over a secret value s to INT. At a later stage, INT is required to hand over s to R and convince R that s is indeed the value which INT received from D.

2.1.2 New Definition, Model, Structure and Properties of ICP

The basic definition of ICP involves only a *single* verifier R [138, 48, 39]. We extend this notion to *multiple* verifiers, specifically to n verifiers/parties denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$ out of which at most t are corrupted by unbounded powerful active adversary. Moreover D and INT are some specific party from $\mathcal{P}$. Thus our

ICP is executed among three entities: a dealer $D \in \mathcal{P}$, an intermediary $INT \in \mathcal{P}$ and the entire set $\mathcal{P}$ acting as verifiers. This will be later helpful in using ICP as a tool in our VSS protocol. Moreover, in contrast to the existing ICPs that deal with single secret, our ICP can deal with *multiple* secrets *concurrently* and thus achieves better communication complexity than multiple execution of ICP dealing with single secret.

The multiple secret, multiple receiver ICP is useful in the design of efficient protocols for statistical VSS and WSS. Statistical VSS is possible iff $n \geq 2t+1$ (provided a physical broadcast channel is available in the system) and for the design of statistical VSS with optimal resilience, we work with $n = 2t+1$. As our ICP is useful in such context, we design our ICP as well with $n = 2t+1$. Thus our ICP can be used for statistical VSS and WSS and they can be used for statistical MPC with optimal resilience (i.e $n = 2t+1$).

2.1.2.1 The Network and Adversary Model

We consider a setting with n parties (we also call them as verifiers) $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ with $n = 2t+1$, that are pairwise connected by a secure (or private) channel. Also $D, INT \in \mathcal{P}$ are two specific parties where D is called as *Dealer* and INT is referred to as *Intermediary*. We further assume that all parties in $\mathcal{P}$ have access to a common broadcast channel. We assume the system to be synchronous. Therefore the protocols operate in a sequence of rounds, where in each round, a party performs some local computation, sends new messages to the other parties through the private channels and broadcasts some information over the broadcast channel, then it receives the messages that were sent by the other parties in this round on the private and broadcast channels.

The adversary that we consider is a *static, threshold, active and rushing* adversary having *unbounded computing power*. The adversary, denoted by $\mathcal{A}_t$, can corrupt at most t parties out of the n parties. Here $\mathcal{A}_t$ may corrupt D as well as INT. The adversary controls and coordinates the actions of the corrupted/faulty parties. We further allow the adversary to be *rushing* [48], i.e. in every round of communication it can wait to hear the messages of the honest parties before sending his own messages. We consider a static adversary, who corrupts all the parties at the beginning of the protocol.

We assume that the messages sent through the channels are from a specified domain. Thus if a party receives a message which is not from the specified domain (or a party receives no message at all), then he replaces it with some pre-defined default message. Thus, we separately do not consider the case when no message or syntactically incorrect message is received by a party.

2.1.2.2 Structure of ICP

As in [138, 39], our ICP is also structured into sequence of following three phases:

1. **Generation Phase**: This phase is initiated by D. Here D hands over the secret S containing ℓ elements from $\mathbb{F}$ (working field of ICP) to *intermediary* INT. In addition, D sends some *authentication information* to INT and some *verification information* to individual verifiers in $\mathcal{P}$.

2. **Verification Phase**: This phase is initiated by INT to acquire an IC Signature on S that will be later accepted by every honest verifier in $\mathcal{P}$. Depending on the behavior of D/INT, secret S OR S along with the *authentication*

information, held by INT at the end of **Verification Phase** will be called as D's IC *signature* on S and will be denoted by $ICSig(D, INT, \mathcal{P}, S)$.

3. **Revelation Phase**: This phase is carried out by INT and the verifiers in $\mathcal{P}$. Here INT reveals $ICSig(D, INT, \mathcal{P}, S)$. The verifiers publish their responses after verifying $ICSig(D, INT, \mathcal{P}, S)$ with respect to their verification information. Depending upon the responses of the verifiers, every verifier $P_i \in \mathcal{P}$ either accepts $ICSig(D, INT, \mathcal{P}, S)$ or rejects it.

2.1.2.3 The properties of ICP

Our ICP satisfies the following properties (which are almost same as the properties, satisfied by the ICP of [138, 48]). In these properties, ϵ denotes the error probability which is negligible (Recall the discussion presented in the beginning of section 1.5 for the meaning of negligible).

1. **ICP-Correctness1:** If D and INT are *honest*, then $ICSig(D, INT, \mathcal{P}, S)$ will be accepted in **Revelation Phase** by each *honest* verifier.

2. **ICP-Correctness2:** If INT is *honest* then at the end of **Verification Phase**, INT possesses an $ICSig(D, INT, \mathcal{P}, S)$, which will be accepted in **Revelation Phase** by all honest verifiers, except with probability ϵ.

3. **ICP-Correctness3:** If D is *honest*, then during **Revelation Phase**, with probability at least $(1 - \epsilon)$, every $ICSig(D, INT, \mathcal{P}, S')$ with $S' \neq S$, produced by a *corrupted* INT will be rejected by *honest* verifiers.

4. **ICP-Secrecy:** If D and INT are *honest* then till the end of **Verification Phase**, S is information theoretically secure from $\mathcal{A}_t$ (that controls t verifiers in $\mathcal{P}$).

2.1.3 The Road-map

In section 2.2, we present our novel ICP with its complete proof. In section 2.3, we compare our ICP with the existing ICPs and show that our ICP attains the best communication and round complexity among all existing ICPs. Section 2.4 introduces several important remarks, facts, definitions and notations for our ICP. Section 2.5 then concentrates on the linearity property of our ICP. Finally, we conclude this chapter in section 2.6.

2.2 A Novel ICP

In this section, we present an ICP called as MVMS-ICP (MVMS stands for Multi Verifier Multi Secret). Protocol MVMS-ICP requires one round for **Generation Phase** and two rounds for **Verification Phase** and **Revelation Phase** each. We will compare MVMS-ICP with the existing ICPs of [138, 48] at the end of this chapter.

To bound the error probability by ϵ, our protocol MVMS-ICP operates over field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n2^{-\kappa}$ and the value of ϵ. Specifically the the *minimum* value of κ that satisfies $\epsilon \geq n2^{-\kappa}$ will determine the field $\mathbb{F}$. Now onwards, similar interpretation has to be drawn in all the subsequent contexts throughout the thesis. Now the relation between

ϵ and κ implies that we have $|\mathbb{F}| \geq \frac{n}{\epsilon}$. Moreover, we have $n = \mathcal{O}(\log \frac{1}{\epsilon})$ (follows from $\epsilon \leq \frac{1}{2^{\alpha n}}$ as mentioned in section 1.5). Now each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{n}{\epsilon}) = \mathcal{O}(\log n + \log \frac{1}{\epsilon}) = \mathcal{O}(\log \frac{1}{\epsilon})$ bits (the last equality in the sequence follows from $n = \mathcal{O}(\log \frac{1}{\epsilon})$).

We now present an informal idea of MVMS-ICP.

The Intuition: In MVMS-ICP, D selects a random polynomial $F(x)$ of degree $\ell + t$, whose lower order ℓ coefficients are the elements of S and delivers $F(x)$ to INT. In addition, D privately delivers to each individual verifier P_i, the value of $F(x)$ at a random, secret *evaluation point* α_i. This distribution of information by D helps to achieve **ICP-Correctness3** property. The reason is that if D is *honest*, then a *corrupted* INT cannot produce an *incorrect* $F'(x) \neq F(x)$ during **Revelation Phase** without being detected by an *honest* verifier with very high probability. This is because a corrupted INT will have no information about the evaluation point of an honest verifier and hence with very high probability, $F'(x)$ will not match with $F(x)$ at the evaluation point held by an honest verifier.

The above distribution by D also maintains **ICP-Secrecy** property. This is because the degree of $F(x)$ is $\ell + t$. But only up to t points on $F(x)$ will be known to $\mathcal{A}_t$ through t corrupted verifiers. Therefore $\mathcal{A}_t$ will fall short by ℓ points to *uniquely* interpolate $F(x)$.

But the above distribution alone is not enough to achieve **ICP-Correctness2**. A *corrupted* D might distribute $F(x)$ to INT and value of some other polynomial (different from $F(x)$) to each honest verifier. To detect this situation, INT and the verifiers interact in *zero knowledge* fashion to check the consistency of $F(x)$ held by INT and the values held by individual verifiers. The specific details of the zero knowledge, along with other formal steps of protocol MVMS-ICP are given in Fig. 2.1.

We now prove the properties of protocol MVMS-ICP.

Claim 2.1 *If D and INT are honest then D will never broadcast S during Ver.*

PROOF: Since INT is honest, he will correctly broadcast $(d, B(x))$ during **Round 1** of Ver. So during **Round 2** of Ver, D will find $B(\alpha_i) = dv_i + r_i$ for all $i = 1, \ldots, n$. Thus D will never broadcast S during Ver. □

Lemma 2.2 (ICP-Correctness1) *If D and INT are honest, then $ICSig(D, INT, \mathcal{P}, S)$ produced by INT during Reveal will be accepted by each honest verifier.*

PROOF: If D is honest, then $(F(x), R(x))$ held by honest INT and (α_i, v_i, r_i) held by honest verifier P_i will satisfy $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$. Moreover by Claim 2.1, D will never broadcast S during Ver. Hence $ICSig(D, INT, \mathcal{P}, S) = F(x)$. Now every honest verifier P_i will broadcast `Accept` in **Round 2** of Reveal as condition **C1** i.e $v_i = F(\alpha_i)$ will hold. Since there are at least $t+1$ honest verifiers, $ICSig(D, INT, \mathcal{P}, S)$ will be accepted by every honest verifier. □

Claim 2.3 *If D is corrupted and $(F(x), R(x))$ held by an honest INT and (α_i, v_i, r_i) held by an honest verifier P_i satisfies $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$, then except with probability $\frac{\epsilon}{n}$, $B(\alpha_i) \neq dv_i + r_i$.*

PROOF: We first prove that for $(F(x), R(x))$ held by an honest INT and (α_i, v_i, r_i) held by honest verifier P_i, there is *only one* non-zero d for which $B(\alpha_i) = dv_i + r_i$,

Figure 2.1: Protocol MVMS-ICP with $n = 2t + 1$ Verifiers

MVMS-ICP$(D, INT, \mathcal{P}, S, \epsilon)$

Gen$(D, INT, \mathcal{P}, S, \epsilon)$: This will take one round

Round 1:

1. D picks and sends the following to INT:
   (a) A random degree-$(\ell + t)$ polynomial $F(x)$ over $\mathbb{F}$, such that the lower order ℓ coefficients of $F(x)$ are elements of S.
   (b) A random degree-$(\ell + t)$ polynomial $R(x)$ over $\mathbb{F}$.
2. D privately sends the following to every verifier P_i:
   (a) (α_i, v_i, r_i), where $\alpha_i \in \mathbb{F} - \{0\}$ is random (all α_i's are distinct), $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$.

Ver$(D, INT, \mathcal{P}, S, \epsilon)$: This will take two rounds

Round 1: INT chooses a random $d \in \mathbb{F} \setminus \{0\}$ and broadcasts $(d, B(x))$ where $B(x) = dF(x) + R(x)$.

Round 2: D checks $dv_i + r_i \stackrel{?}{=} B(\alpha_i)$ for $i = 1, \ldots, n$. If D finds any inconsistency, he broadcasts S.

If D has broadcasted S, then $ICSig(D, INT, \mathcal{P}, S) = S$, else $ICSig(D, INT, \mathcal{P}, S) = F(x)$.

Reveal$(D, INT, \mathcal{P}, S, \epsilon)$: This will take two rounds

Round 1 INT broadcasts $ICSig(D, INT, \mathcal{P}, S)$ (i.e either $F(x)$ or S).

Round 2: Verifier P_i broadcasts `Accept` in the following conditions.

1. If $ICSig(D, INT, \mathcal{P}, S) = S$, then if the S broadcasted by D in **Round 2** of Ver is same as $ICSig(D, INT, \mathcal{P}, S)$.
2. If $ICSig(D, INT, \mathcal{P}, S) = F(x)$, then if one of the following conditions holds.
   (a) **C1:** $v_i = F(\alpha_i)$; OR
   (b) **C2:** $B(\alpha_i) \neq dv_i + r_i$ ($B(x)$ was broadcasted by INT during Ver) and D did not broadcast S in **Round 2** of Ver.

Otherwise, P_i broadcasts `Reject`.

Local Computation (By Every Verifier) : If at least $(t + 1)$ verifiers have broadcasted `Accept` during **Round 2** of Reveal then accept $ICSig(D, INT, \mathcal{P}, S)$. Else reject $ICSig(D, INT, \mathcal{P}, S)$.

even though $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. For otherwise, assume there exists another non-zero element $e \neq d$, for which $B(\alpha_i) = ev_i + r_i$ is true, even if $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. This implies that $(d - e)F(\alpha_i) = (d - e)v_i$ or $F(\alpha_i) = v_i$, which is a contradiction. Now since d is randomly chosen by honest INT *only after* D handed over $(F(x), R(x))$ to INT and (α_i, v_i, r_i) to P_i, a corrupted D has to guess d in advance during Gen to make sure that $B(\alpha_i) = dv_i + r_i$ holds. However, D can guess d with probability at most $\frac{1}{|\mathbb{F}|-1} \approx \frac{\epsilon}{n}$. Hence only with probability at most $\frac{\epsilon}{n}$, corrupted D can ensure $B(\alpha_i) = dv_i + r_i$, even though $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. □

Lemma 2.4 (ICP-Correctness2) *If INT is honest then at the end of* Ver*, INT possesses an $ICSig(D, INT, \mathcal{P}, S)$, which will be accepted in* Reveal *by all honest verifiers, except with probability ϵ.*

Proof: We consider the case when D is corrupted, because when D is honest, the lemma follows from Lemma 2.2. Now the proof can be divided into following two cases:

1. $ICSig(D, INT, \mathcal{P}, S) = S$: This implies that D has broadcasted S during **Round 2** of Ver. In this case, the lemma holds trivially, without any error. This is because the honest INT will correctly broadcast $ICSig(D, INT, \mathcal{P}, S) = S$ during **Round 1** of Reveal and every honest verifier will find that S broadcasted by INT is same as the one that was broadcasted by D during **Round 2** of Ver. So all honest verifiers (at least $t+1$) will broadcast `Accept` and hence $ICSig(D, INT, \mathcal{P}, S)$ will be accepted by all honest verifiers.

2. $ICSig(D, INT, \mathcal{P}, S) = F(x)$: This implies that D has not broadcasted anything during **Round 2** of Ver. Here, we first show that except with probability $\frac{\epsilon}{n}$, each honest verifier will broadcast `Accept` during Reveal. So let P_i be an honest verifier. We have now the following cases depending on the relation that holds between the information held by INT (i.e $(F(x), R(x))$) and information held by the honest P_i (i.e (α_i, v_i, r_i)):

    (a) *If $F(\alpha_i) = v_i$*: Here P_i will broadcast `Accept` without any error probability as condition **C1** (i.e $F(\alpha_i) = v_i$) will hold.

    (b) *If $F(\alpha_i) \neq v_i$ and $R(\alpha_i) = r_i$*: Here P_i will broadcast `Accept` without any error probability, as condition **C2** (i.e $B(\alpha_i) \neq dv_i + r_i$) will hold.

    (c) *If $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$*: Here P_i will broadcast `Accept` except with probability $\frac{\epsilon}{n}$, as condition **C2** will hold, except with probability $\frac{\epsilon}{n}$ (see Claim 2.3).

    As shown above, there is a negligible error probability of $\frac{\epsilon}{n}$ with which an *honest* P_i may broadcast Reject when $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$ (i.e the third case). This happens if a corrupted D can guess the unique d in Gen, corresponding to P_i and it so happens that INT also selects the same d in Ver and therefore condition **C2** does not hold good for P_i in Reveal. Now D can guess a d_i for each honest verifier P_i and if it so happens that honest INT chooses d which is same as one of those $t+1$ d_i's guessed by D, then condition **C2** will not be satisfied for the honest verifier P_i for whom $d_i = d$ and therefore P_i will broadcast `Reject`. This may lead to the rejection of $ICSig(D, INT, \mathcal{P}, S)$, as t corrupted verifiers may always broadcast `Reject`. But the above event can happen with error probability $\frac{t+1}{|\mathbb{F}|-1} = (t+1)\frac{\epsilon}{n} \approx \epsilon$. This is because there are $t+1$ d_i's and INT has selected some d randomly from $\mathbb{F} \setminus \{0\}$. This implies that all honest verifiers will broadcast `Accept` during Reveal, except with error probability ϵ.

This completes the proof of the lemma. □

Lemma 2.5 (ICP-Correctness3) *If D is honest then during* Reveal*, with probability at least $1-\epsilon$, every $ICSig(D, INT, \mathcal{P}, S')$ with $S' \neq S$ revealed by a corrupted INT will be rejected by honest verifiers.*

PROOF: Here again we have the following two cases:

1. $ICSig(D, INT, \mathcal{P}, S) = S$: This implies that D has broadcasted S during **Round 2** of Ver. In this case if a corrupted INT tries to reveal $ICSig(D, INT, \mathcal{P}, S')$ where $S' \neq S$ then all honest verifiers (at least $t+1$) will broadcast `Reject` during Reveal. This is because the honest verifiers will find that S' is not same as S which was broadcasted by D during **Round 2** of Ver.

2. $ICSig(D, INT, \mathcal{P}, S) = F(x)$: This implies that D has not broadcasted anything during **Round 2** of Ver. Here a corrupted INT can produce $S' \neq S$ by broadcasting $F'(x) \neq F(x)$ during Reveal such that the lower order ℓ coefficients of $F'(x)$ is S'. We now claim that if INT does so, then except with probability $\frac{\epsilon}{n}$, an honest verifier P_i will A-cast `Reject` during Reveal. In the following, we show that the conditions for which the honest verifier P_i would broadcast `Accept` are either impossible or may happen with probability $\frac{\epsilon}{n}$:

   (a) $F'(\alpha_i) = v_i$: Since P_i and D are honest, corrupted INT has no information about α_i, v_i. Hence the probability that INT can ensure $F'(\alpha_i) = v_i = F(\alpha_i)$ is same as the probability with which INT can correctly guess α_i, which is at most $\frac{1}{|\mathbb{F}-1|} \approx \frac{\epsilon}{n}$ (since α_i is randomly chosen by D from $\mathbb{F}$).

   (b) $B(\alpha_i) \neq dv_i + r_i$: This case is never possible because D is honest. If $B(\alpha_i) \neq dv_i + r_i$ corresponding to P_i, then honest D would have broadcasted S during **Round 2** of Ver and hence $ICSig(D, INT, \mathcal{P}, S)$ would have been equal to S, which is a contradiction to our assumption that $ICSig(D, INT, \mathcal{P}, S) = F(x)$.

   As shown above, there is a negligible error probability of $\frac{\epsilon}{n}$ with which an honest P_i may broadcast Accept, even if the corrupted INT produces $F'(x) \neq F(x)$. This happens if the corrupted INT can guess α_i corresponding to honest verifier P_i. Now there are $t+1$ honest verifiers. A corrupted INT can guess α_i for any one of those $t+1$ honest verifiers and thereby can ensure that $F'(\alpha_i) = v_i$ holds for some honest P_i (which in turn implies P_i will broadcast Accept). This will ensure that INT's $ICSig(D, INT, \mathcal{P}, S')$ will be accepted, as t corrupted verifiers may always broadcast Accept. But the above event can happen with probability at most $\frac{t+1}{|\mathbb{F}|-1} = (t+1)\frac{\epsilon}{n} \approx \epsilon$. This asserts that every $ICSig(D, INT, \mathcal{P}, S')$ with $S' \neq S$, revealed by a corrupted INT will be rejected by all honest verifiers with probability at least $(1-\epsilon)$. □

Lemma 2.6 (ICP-Secrecy) *If D and INT are honest, then till the end of* Ver*, S is information theoretically secure from $\mathcal{A}_t$ (that controls t verifiers in $\mathcal{P}$).*

PROOF: During Gen, $\mathcal{A}_t$ will know t distinct points on $F(x)$ and $R(x)$. Since both $F(x)$ and $R(x)$ are of degree-$(\ell+t)$, the lower order ℓ coefficients of both

$F(x)$ and $R(x)$ are information theoretically secure. During Ver, $\mathcal{A}_t$ will know d and $dF(x) + R(x)$. Since both $F(x)$ and $R(x)$ are random and independent of each other, the lower order ℓ coefficients of $F(x)$ remain to be information theoretically secure. Also, if D and INT are honest, then D will never broadcast S during Ver (from Claim 2.1). Hence the lemma. □

Theorem 2.7 *Protocol MVMS-ICP is an efficient ICP.*

PROOF: Follows from Lemma 2.2, 2.4, 2.5 and 2.6. □

Theorem 2.8 (Round Complexity of MVMS-ICP) *In protocol MVMS-ICP, Gen requires one round, Ver and Reveal requires two rounds each.*

PROOF: Follows from the protocol description as presented in Fig. 2.1 □

Theorem 2.9 (Communication Complexity of MVMS-ICP) *Protocol MVMS-ICP attains the following bounds:*

- *Protocol Gen privately communicates $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$ bits.*
- *Protocol Ver and Reveal requires broadcast of $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$ bits each.*

PROOF: In protocol Gen, D privately gives $\ell + t$ field elements to INT and three field elements to each verifier. Since each field element can be represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits, Gen incurs a private communication of $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$ bits. In protocol Ver, INT broadcasts $B(x)$ containing $\ell + t$ field elements, thus incurring broadcast of $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$ bits. Moreover, D may broadcast S which will incur broadcast of $\mathcal{O}(\ell \log \frac{1}{\epsilon})$ bits. Therefore, in total Ver requires broadcast of $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$ bits. In protocol Reveal, INT broadcasts $F(x)$, consisting of $\ell + t$ field elements, while each verifier broadcasts `Accept/Reject` signal. So Reveal involves broadcast of $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$ bits. □

In the next section, we compare our protocol MVMS-ICP with the existing ICPs with respect to communication and round complexity and show that MVMS-ICP is the most communication and round efficient ICP in the literature. After that, we describe few important remarks, facts and notations. Lastly, we explore an important property of MVMS-ICP called linearity property. Linearity of MVMS-ICP will be used in Chapter 5 for constructing our statistical MPC protocol with $n = 2t + 1$.

2.3 Comparison of MVMS-ICP with the ICPs of [138] and [48]

Both the ICPs of [138] and [48] are designed in single verifier and single secret model. But they can be extended to the case of multiple (i.e. n) verifiers easily. Indeed in [138, 48], the single verifier ICPs were executed in parallel for n verifiers in the implementation of VSS protocols. Moreover, as the protocols were designed for single secret, they can be extended for ℓ secrets by ℓ parallel invocations of the protocols. Since protocol MVMS-ICP is designed to handle n verifiers and ℓ secrets concurrently, in Table 2.1, we compare our MVMS-ICP with the ICPs of [138] and [48] extended for n verifiers and ℓ secrets.

Table 2.1: Communication Complexity and Round Complexity of protocol MVMS-ICP and Existing ICP with $n = 2t+1$ verifiers and ℓ secrets.

	Communication Complexity in Bits			Round Complexity		
Ref.	Gen	Ver	Reveal	Gen	Ver	Reveal
[138]	Private– $\mathcal{O}(\ell n(\log \frac{1}{\epsilon})^2)$	Broadcast– $\mathcal{O}(\ell n(\log \frac{1}{\epsilon})^2)$	Broadcast– $\mathcal{O}(\ell n(\log \frac{1}{\epsilon})^2)$	1	at least 3	2
[48]	Private– $\mathcal{O}(\ell n \log \frac{1}{\epsilon})$	Broadcast– $\mathcal{O}(\ell n \log \frac{1}{\epsilon})$	Broadcast– $\mathcal{O}(\ell n \log \frac{1}{\epsilon})$	1	3	2
This thesis MVMS-ICP	Private– $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$	Broadcast– $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$	Broadcast– $\mathcal{O}((\ell + n) \log \frac{1}{\epsilon})$	1	2	2

2.4 Some Important Remarks, Facts, Definitions and Notations

2.4.1 MVMS-ICP with One Round of Reveal

It is interesting to note that if we restrict the adversary $\mathcal{A}_t$ to a non-rushing adversary then the two rounds of Reveal can be collapsed into a single round where INT broadcasts $ICSig(D, INT, \mathcal{P}, S)$ and simultaneously every verifiers broadcast their values (α_i, v_i, r_i). It is easy to check that all the properties of ICP will hold in such a case. But in the presence of rushing adversary, the two rounds are needed in order to force a corrupted INT to commit to the polynomial $F(x)$ prior to seeing the evaluation points, as this knowledge can enable the adversary to publish a polynomial that can match with the values broadcasted by the honest verifiers, which would violate the **ICP-Correctness3** property of the protocol. However, if the adversary is non-rushing then this property is achieved via the synchronicity of the step. Hence, we have the following theorem:

Theorem 2.10 *If the adversary is non-rushing then there exists an efficient ICP with one round in Gen, two rounds in Ver and one round in Reveal.*

Later we will show that due to this theorem, all the VSS and WSS protocols that use our ICP as building block can be designed with one round of reconstruction when the adversary is assumed to be non-rushing.

2.4.2 MVMS-ICP with Single Secret and $n = 3t + 1$ Verifiers

Here we disclose two important facts about protocol MVMS-ICP, which will be required for our subsequent chapters:

1. Though MVMS-ICP has been designed to deal with ℓ secrets concurrently, we may use it for single secret when necessary. We may reflect this by putting $\ell = 1$ in all places. In fact in Chapter 3, we have used MVMS-ICP for single secret to keep the description simple (also because Chapter 3 deals with only round complexity and communication complexity plays a less significant role there).

2. Though MVMS-ICP has been designed for $n = 2t+1$ verifiers, it achieves all its properties even when $n = 3t + 1$. In Chapter 6, we will use MVMS-ICP with $n = 3t + 1$ verifiers.

2.4.3 A Definition

We now present the following definition:

Definition 2.11 (IC Signature with ϵ Error) *An IC signature $ICSig(D, INT, \mathcal{P}, S)$ for some secret S, is said to have ϵ error, if it satisfies the following:*

1. **ICP-Correctness1** *without any error;*
2. **ICP-Correctness2** *with error probability of at most ϵ;*
3. **ICP-Correctness3** *with error probability of at most ϵ;*
4. **ICP-Secrecy** *without any error.*

Notice that if an IC signature is generated in MVMS-ICP (which is executed with error parameter ϵ), then the IC signature will have ϵ error. This follows from the proofs of Lemma 2.2, 2.4, 2.5 and 2.6.

2.4.4 Notation for using MVMS-ICP

Notation 2.12 (Notation for Using MVMS-ICP) *We say that:*

1. *"D sends $ICSig(D, INT, \mathcal{P}, S)$ having ϵ error to INT " to mean that D executes Gen$(D, INT, \mathcal{P}, S, \epsilon)$;*
2. *"INT receives $ICSig(D, INT, \mathcal{P}, S)$ having ϵ error from D" to mean that the parties have executed Ver$(D, INT, \mathcal{P}, S, \epsilon)$;*
3. *"INT reveals $ICSig(D, INT, \mathcal{P}, S)$ having ϵ error" to mean that Reveal$(D, INT, \mathcal{P}, S, \epsilon)$ has been executed.*

Clearly if D sends $ICSig(D, INT, \mathcal{P}, S)$ to INT in i^{th} round, then INT will receive $ICSig(D, INT, \mathcal{P}, S)$ in $(i + 2)^{th}$ round, as Ver requires two rounds.

2.5 Linearity of Protocol MVMS-ICP

The IC signature generated in MVMS-ICP satisfies **linearity** property, which is exploited heavily in our VSS and MPC protocol in Chapter 4 and 5. Specifically, consider the following settings: let in q different instances of MVMS-ICP, D has handed over IC Signature on q different set of ℓ secrets to INT, namely $S_i = (s_i^1, \ldots, s_i^\ell)$, for $i = 1, \ldots, q$. Moreover, let *D has used the same α_i as secret evaluation point for verifier P_i in all the q instances of MVMS-ICP* (an honest D can always ensure it). This condition on α_i is very important and we refer this as *the condition for linearity of IC signatures*. Though linearity property accounts for any form of linear function, we will demonstrate the linearity property with respect to addition operation. This is because addition/subtraction are the linear functions that will be used in our VSS and MPC protocols. So let $S = S_1 + \ldots + S_q$, where $S = (s^1, \ldots, s^\ell)$ and $s^l = s_1^l + \ldots + s_q^l$, for $l = 1, \ldots, \ell$. Now INT can compute $ICSig(D, INT, \mathcal{P}, S)$ using $ICSig(D, INT, \mathcal{P}, S_i)$ for $i = 1, \ldots, q$ and the verifiers can compute verification information corresponding to $ICSig(D, INT, \mathcal{P}, S)$, without doing any further communication. For the sake of completeness, we present a protocol in Fig. 2.2 showing how INT

and verifiers can achieve the above. Informally in the protocol we use the linearity property of polynomials. That is, if $ICSig(D, INT, \mathcal{P}, S_1) = F_1(x)$ and $ICSig(D, INT, \mathcal{P}, S_2) = F_2(x)$, then $ICSig(D, INT, \mathcal{P}, S_1+S_2) = F_1(x)+F_2(x)$. Similarly, if $F_1(\alpha_i)$ and $F_2(\alpha_i)$ are the verification information of verifier P_i corresponding to $ICSig(D, INT, \mathcal{P}, S_1)$ and $ICSig(D, INT, \mathcal{P}, S_2)$ respectively, then $F_1(\alpha_i) + F_2(\alpha_i)$ will be the verification information of verifier P_i corresponding to $ICSig(D, INT, \mathcal{P}, S_1 + S_2)$.

In the protocol, it might be possible that some $ICSig(D, INT, \mathcal{P}, S_i)$ is a polynomial of degree $\ell + t$ (this implies that D has not broadcasted anything during Ver of i^{th} signature giving instance), while some other $ICSig(D, INT, \mathcal{P}, S_j)$ is S_j (this implies that D has broadcasted S_j during Ver of j^{th} signature giving instance). In such a case, INT finds a $\ell + t$ degree polynomial $F_j(x)$, whose lower order ℓ coefficients are elements of S_j and the remaining coefficients are some publicly known default values and assumes the polynomial to be $ICSig(D, INT, \mathcal{P}, S_j)$. Notice that such $F_j(x)$ will be known publicly, as S_j is broadcasted by D. Accordingly, every verifier P_i considers $F_j(\alpha_i)$ as his verification information corresponding to $ICSig(D, INT, \mathcal{P}, S_j)$. Once this is done then all the q IC signatures will be $\ell + t$ degree polynomials and hence INT can use the linearity property of the polynomials (as explained above) to compute the addition of IC signatures.

Now we show that a linearly combined IC signature that is computed from q IC signatures (using protocol in Fig. 2.2), each having ϵ error, will have ϵ error. For this, we prove the following lemma:

Lemma 2.13 *Assuming each of the q individual IC signatures, $ICSig(D, INT, \mathcal{P}, S_j)$ has ϵ error, the linearly combined IC signature, $ICSig(D, INT, \mathcal{P}, S)$ will also have ϵ error.*

Proof: We will examine each of the four properties of IC signature one by one depending on whether D and/or INT are honest or corrupted. When D and INT are honest, then it is easy to see that $ICSig(D, INT, \mathcal{P}, S)$ will abide by **ICP-Correctness1** and **ICP-Secrecy** without any error.

Now when D is honest and INT is corrupted, $ICSig(D, INT, \mathcal{P}, S)$ satisfies **ICP-Correctness3** with error probability ϵ, which is same as the error of individual IC signatures. This is because, here the error probability depends on correctly guessing one of the honest P_i's α_i (recall that same α_i is associated with P_i corresponding to all the individual IC signatures).

Finally, we show that when D is corrupted and INT is honest, $ICSig(D, INT, \mathcal{P}, S)$ satisfies **ICP-Correctness2** with error probability ϵ. The worst case that causes this error probability is as follows:

1. To every honest verifier P_i, D gives $v_{ji} \neq F_j(\alpha_i)$ and $r_{ji} \neq R_j(\alpha_i)$, corresponding to exactly one $j \in \{1, \ldots, q\}$;

2. For all other $j \in \{1, \ldots, q\}$, D gives $v_{ji} = F_j(\alpha_i)$ and $r_{ji} = R_j(\alpha_i)$ to every honest verifier P_i.

In this case, from the proof of Lemma 2.4, $B_j(\alpha_i) \neq d_j v_{ji} + d_j r_{ji}$ will not hold for some honest P_i, except with probability ϵ. Now notice that if D delivers v_{ji}, r_{ji} satisfying $v_{ji} \neq F_j(\alpha_i)$ and $r_{ji} \neq R_j(\alpha_i)$ for more j's, then D has to guess more d_j's and hence the probability with which D can guess all those d_j's will

Figure 2.2: Linearity of Protocol MVMS-ICP Over Addition Operation.

Assumption:

1. D has sent $ICSig(D, INT, \mathcal{P}, S_j)$ having ϵ error to INT, for $j = 1, \ldots, q$, where $S_j = (s_j^1, \ldots, s_j^\ell)$. Let D has used the same α_i as secret evaluation point for verifier P_i in all the q instances for giving IC signatures. Moreover, let INT has used random value d_j in **Round 1** of Ver for j^{th} signature giving instance of MVMS-ICP.

2. INT has received $ICSig(D, INT, \mathcal{P}, S_j)$ having ϵ error from D.

3. For every $j \in \{1, \ldots, q\}$, such that $ICSig(D, INT, \mathcal{P}, S_j)$ is a polynomial of degree $\ell + t$, let $ICSig(D, INT, \mathcal{P}, S_j) = F_j(x)$, i.e D had used $F_j(x)$ to hide S_j. Moreover let P_i has the verification information v_{ji}, which is supposed to be same as $F_j(\alpha_i)$.

Local Computation to Compute Addition of IC Signatures:

1. For all $j \in \{1, \ldots, q\}$, such that $ICSig(D, INT, \mathcal{P}, S_j) = S_j$, INT assumes a degree $\ell + t$ polynomial $F_j(x)$ whose lower order ℓ coefficients are the elements of S_j and the remaining coefficients are some publicly known default values. Notice that such $F_j(x)$ polynomials will be known publicly. For every such $F_j(x)$, verifier P_i computes his verification information as $v_{ji} = F_j(\alpha_i)$.

2. Now to compute $ICSig(D, INT, \mathcal{P}, S)$, INT sets $F(x) = \sum_{j=1}^{q} F_j(x)$ and assigns $ICSig(D, INT, \mathcal{P}, S) = F(x)$.

3. Every verifier P_i computes his verification information corresponding to $ICSig(D, INT, \mathcal{P}, S)$ in the following way: $v_i = \sum_{j=1}^{q} v_{ji}$.

Revelation of Linear IC Signature:

1. INT broadcasts $ICSig(D, INT, \mathcal{P}, S)$ (i.e $F(x)$).

2. Verifier P_i broadcasts `Accept` if one of the following conditions holds.

   (a) **C1:** $v_i = F(\alpha_i)$; OR

   (b) **C2:** For some $j \in \{1, \ldots, q\}$, $B_j(\alpha_i) \neq d_j v_{ji} + r_{ji}$ ($B_j(x)$ was broadcasted by INT during **Round 1** of Ver of j^{th} signature giving instance) and D has not broadcasted S_j in **Round 2** of Ver of j^{th} signature giving instance.

   Otherwise, P_i broadcasts `Reject`.

Local Computation (By Every Verifier): If at least $(t + 1)$ verifiers have broadcasted `Accept` then accept $ICSig(D, INT, \mathcal{P}, S)$ and hence S. Else reject $ICSig(D, INT, \mathcal{P}, S)$.

decrease beyond ϵ. Hence we proved that when D is corrupted and INT is honest, $ICSig(D, INT, \mathcal{P}, S)$ satisfies **ICP-Correctness2** with error probability ϵ.

Hence the lemma. $\square$

The linearity of IC signatures also captures the following case: Let in an execution of MVMS-ICP, D has handed over IC Signature on a set of ℓ secrets to INT, say $b^1, \ldots, b^\ell$. That is at the end of Ver, INT holds $ICSig(D, INT, \mathcal{P}, (b^1, \ldots, b^\ell))$. Also let $(a^1, \ldots, a^\ell)$ are some publicly known values. Now INT can compute $ICSig(D, INT, \mathcal{P}, (b^1 - a^1, \ldots, b^\ell - a^\ell))$ and similarly verifiers can update their verification information accordingly, by doing local computation. Later in Reveal, INT can reveal $ICSig(D, INT, \mathcal{P}, (b^1 - a^1, \ldots, b^\ell - a^\ell))$ to the verifiers. Moreover, the above idea can be extended for any number of IC signatures and any number of sets containing publicly known values. In Chapter 5, we will need all the above concepts extensively. Now we present the following important notes.

Note 2.14 *We would like to alert that linearity of IC signatures holds only when all the IC signatures are generated by same party, say P (who acts as a dealer). Moreover, P should abide by the condition for the linearity of IC signatures. Linearity does not hold on the IC signatures that are generated by different parties, as they will not satisfy condition for the linearity of IC signatures (because different parties may choose different α_i for verifier P_i in their signature giving instance).*

Note 2.15 *Many protocols in the subsequent chapters will use MVMS-ICP as a black box directly or indirectly with different error probability. To bound the error probability by ϵ, different protocols will invoke (directly or indirectly through some other sub-protocol) MVMS-ICP with different error probability. Consequently, depending on the* **minimum** *error probability with which MVMS-ICP is invoked in a protocol will determine the exact relationship between ϵ and κ for that protocol (which in term determine the field $\mathbb{F} = GF(2^\kappa)$ for that protocol).*

2.6 Conclusion and Open Problems

In this chapter, we have extended the basic bare-bone definition of ICP, introduced by Rabin et al. [138] and subsequently followed by [39, 48], to capture multiple verifiers and multiple secrets concurrently. Then we have presented a novel ICP (matching with our definition) that turns out to be the best ICP in the literature as per the round and communication complexity. We then explored the linearity property of our ICP (will be used in Chapter 5). We now conclude this chapter with an interesting open question:

Open Problem 1 *Can we improve the round and communication complexity of MVMS-ICP when there are $n = 2t + 1$ verifiers?*

This leads to a more general question:

Open Problem 2 *What is the round and communication complexity lower bound for ICP with $n = 2t + 1$ verifiers?*

Chapter 3

The Round Complexity of Statistical VSS and WSS

The round complexity of interactive protocols is one of their most important complexity measures. The investigation of the round complexity of protocols is usually conducted under the assumption that the protocols are error-free. In this chapter, we investigate the round complexity of VSS and WSS (a weaker notion of VSS) by introducing a probability of error and examine the question of whether introducing a probability of error into the executions can improve on known results and lower bounds. In fact, our results in this chapter show that existing lower bounds for the round complexity of perfect VSS and WSS can be circumvented by introducing a negligible probability of error.

3.1 Introduction

3.1.1 Relevant Literature of VSS

Due to the central importance of VSS in the context of many cryptographic protocols such as MPC [3, 19, 5, 6, 7, 20, 12, 13, 14, 21, 9, 36, 41, 48, 49, 52, 95, 93, 98, 101, 103, 104, 135, 138, 143, 126], BA [68, 18, 29, 39, 35, 118, 72, 110, 2, 24, 25, 26, 30, 31, 32, 44, 56, 54, 57, 59, 60, 61, 74, 70, 71, 65, 67, 78, 86, 89, 114, 117, 134, 136, 150, 148, 149], etc, the VSS problem has drawn much attention over the years (e.g. [43, 55, 108, 9, 95, 20, 41, 62, 63, 137, 48, 21, 39, 138, 73, 91, 93, 109, 125, 12, 14, 98, 126, 50, 47, 35, 96, 28, 133, 66, 64, 8, 37, 22, 53, 92, 123, 145, 34, 97]) and many aspects of the problem have been studied.

In information theoretic settings (i.e. under the assumption of a *computationally unbounded adversary*), there are mainly two flavors of VSS: Perfect VSS (i.e. error free) and statistical VSS (involves some probability of error). It is well known that perfect VSS is possible iff $n \geq 3t+1$ [55]. On the other hand, statistical VSS where a probability of error is allowed, is achievable for $n \geq 2t+1$, assuming availability of a public broadcast channel [138] (in addition to the point-to-point secure channels between every pair of parties).

The study of the round complexity of VSS in the information theoretic security setting, was initiated by Gennaro et al. [91]. Their investigation was conducted for perfect VSS i.e under the assumption that the protocols are error-free. They refer to the round complexity of VSS as the number of rounds in the sharing phase and prove that a 3-round sharing error-free VSS is possible only if $n \geq 3t+1$, and match it with an inefficient upper bound. Fitzi et al. [73] present an efficient

3-round sharing VSS protocol in this setting. The protocol of Fitzi et al. used the broadcast channel in more than one round of the sharing phase and Katz et al. [109] showed how to achieve the same result while using a single round of broadcast. Apart from the lower bound for 3-round sharing VSS, [91] also reports that 1-round sharing VSS is possible iff $t = 1$ and $n \geq 5$ and 2-round sharing VSS is possible iff $n \geq 4t + 1$. In summary, the results reported in [91, 73, 109] are presented in Table 3.1.

Table 3.1: Summary of VSS Bounds and Round Complexity.

# Sharing Rounds	Conditions	Comment	Efficient Protocol?
1	$t = 1$; $n \geq 5$	iff	Yes
2	$n \geq 4t + 1$	iff	Yes
3	$n \geq 3t + 1$	iff	Yes

So far in the literature, there are three statistical VSS protocols with $n = 2t+1$ [138, 48, 49]. All of them require fairly very high round complexity.

3.1.2 Our Results on Statistical VSS

In this chapter, we examine the round complexity of statistical VSS and also investigate the question of whether the lower bounds for the round complexity of VSS can be overcome by introducing a negligible probability of error. In our work, we show that if we allow negligible error probability then there exists:

1. An efficient 1-round sharing, 2-round reconstruction VSS protocol for $t = 1$ and $n \geq 4$.
2. An efficient 2-round sharing, 2-round reconstruction VSS protocol for $n \geq 3t + 1$.
3. An efficient 3-round sharing, 2-round reconstruction VSS protocol for $t = 1$ and $n \geq 3$.
4. An *in-efficient* 4-round sharing, 2-round reconstruction VSS protocol for $n \geq 2t + 1$.
5. An efficient 5-round sharing, 2-round reconstruction VSS protocol for $n \geq 2t + 1$.

Interestingly, in all the above protocols, 1-round reconstruction is possible if we assume the adversary to be non-rushing, where a rushing adversary can wait to hear the incoming messages in a given round prior to sending out its own messages. The VSS protocols mentioned in 3^{rd}, 4^{th} and 5^{th} items require optimal number of rounds in reconstruction phase. This is because in [49], it is proved that VSS with $n = 2t + 1$ and any value of $t \geq 1$ will require 2 rounds in reconstruction when the adversary is rushing and the reconstruction can be collapsed in a single round when the adversary is considered to be non-rushing. We do not know whether the same holds for our 1-round sharing and 2-round sharing VSS protocols (mentioned in 1^{st} and 2^{nd} items). But here we prove that our protocols mentioned in 1^{st} and 2^{nd} items are optimal both in resilience and number of sharing rounds by showing:

1. There does not exist any 1-round sharing statistical VSS protocol for ($t \geq 2$; $n \geq 4$) and ($t = 1$; $n < 4$).

2. There does not exist any 2-round sharing statistical WSS (and hence VSS) for $n \leq 3t$.

In summary, our results are presented in Table 3.2.

Table 3.2: Summary of Statistical VSS Bounds and Round Complexity.

# Sharing Rounds	Conditions	Comment	Efficient Protocol?
1	$t = 1$; $n \geq 4$	iff	Yes
2	$n \geq 3t + 1$	iff	Yes
3	$t = 1$; $n \geq 3$	only if	Yes
4	$n \geq 2t + 1$	only if	No
5	$n \geq 2t + 1$	only if	Yes

Our results for 2-round sharing statistical VSS show that existing lower bounds of [91] for 3-round sharing error-free VSS can be circumvented by introducing a negligible probability of error. Apart from this, our results for 2-round sharing VSS matches the sharing phase round complexity of the best known protocols in the computational setting [64, 133] with no set-up assumptions (but note that these protocols use a one round reconstruction phase). Our VSS protocol for $n \geq 3t + 1$ also achieve the design optimization of Katz et al. [109] and use a single round of broadcast in the sharing phase and no broadcasts at all in the reconstruction phase. Finally, comparing Table 3.1 and 3.2, we see that introducing error probability in VSS protocol also helps to increase the fault tolerance of VSS.

3.1.3 Our Results on Statistical WSS

Generally, WSS is used as a tool to design VSS protocols [138, 137]. Informally WSS is a primitive which satisfies the same properties as VSS except for the commitment property. VSS has a *strong commitment*, which requires that at the end of the sharing, there is a fixed value s^* and that the honest parties output this value in the reconstruction phase. In contrast, WSS has a *weaker commitment* property which requires that at the end of the reconstruction phase, the honest parties output s^* or NULL.

The study of the round complexity of perfect WSS in the information theoretic security setting, was initiated in [73]. In [73], the authors referred to the round complexity of WSS as the number of rounds in the sharing phase and have shown that

1. Efficient 1-round as well as 2-round sharing WSS protocol is possible iff $n \geq 4t + 1$.

2. Efficient 3-round sharing WSS protocol is possible iff $n \geq 3t + 1$.

In this chapter, we completely resolve the round complexity of WSS involving negligible error probability by showing that:

1. Efficient 1-round as well as 2-round sharing WSS protocol is possible iff $n \geq 3t + 1$.
2. Efficient 3-round sharing WSS protocol is possible iff $n \geq 2t + 1$.

Our results clearly show that probabilistically relaxing the conditions of WSS helps to increase the fault tolerance. The 2-round sharing WSS presented in our chapter is used to build our 2-round sharing VSS protocol.

3.1.4 The Working Field of Our Protocols

All our protocols (VSS and WSS) involve an error probability of ϵ. To bound the error probability by ϵ, our protocols operate with values from a finite field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the value of ϵ and the relation between ϵ and κ. The exact relationship between ϵ and κ is different for different protocols and therefore it is mentioned in respective sections. In all cases, each element of $\mathbb{F}$ can be represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits. We say that our protocols are efficient if the communication and computation of the honest parties are polynomial in κ or $\log \frac{1}{\epsilon}$. All the protocols, except 4-round sharing VSS protocol, presented in this chapter perform computation and communication which are poly(κ) or poly($\log \frac{1}{\epsilon}$).

3.1.5 On the Definition of Round Complexity of VSS and WSS

As we have stated earlier, the common definition for the round complexity of VSS is the number of rounds in the sharing phase. This is a natural definition for the perfect (i.e., zero error) setting, as the reconstruction can always be done in one round (by having all parties reveal their complete view generated at the end of sharing phase). However, in all our protocols we have a reconstruction phase that cannot be collapsed into a single round. In [49], it is proved that VSS with $n = 2t+1$ and $t \geq 1$ will require two rounds in reconstruction when the adversary is rushing and the reconstruction can be collapsed in a single round when the adversary is considered to be non-rushing. But we do not know whether the same holds with respect to our 1-round sharing statistical VSS (with $n \geq 4$ and $t = 1$) and 2-round sharing statistical VSS (with $n \geq 3t + 1$) protocols. This indicates that a different definition for the round complexity of VSS may be needed, which is the total number of rounds in the sharing plus the number of rounds in the reconstruction. It is to be noted that the previous 3-round sharing *perfect* VSS [91, 73, 109] and our result for 2-round sharing 2-round reconstruction statistical VSS exhibit VSS with a *total* of four rounds[1]. This introduces the question of what is the lower bound on the total number of rounds for VSS with $n = 3t + 1$ or with $n = 2t + 1$. We may as well ask similar questions for WSS. Even all the three WSS protocols presented in this chapter require 2-round reconstruction and similar to the case of VSS, they can be collapsed to single round when the adversary is non-rushing.

[1]As the total number of rounds in both protocols is the same, the question of which protocol to use depends on the application. For applications where there is a need of more efficiency during the sharing, i.e. fewer number of rounds, our two round sharing statistical protocol should be used.

3.1.6 The Network and Adversary Model

We follow the network model of [138, 91]. The model is presented in Section 2.1.2 of Chapter 2. Recall that the set of parties is denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$ and t out of the n parties can be under the influence of a *computationally unbounded powerful, Byzantine (active), rushing adversary*, denoted as $\mathcal{A}_t$. Apart from pairwise secure channels, there is a physical broadcast channel available in the network. Here n takes different values (such as $3t + 1$ and $2t + 1$) in different protocols.

3.1.7 Definitions of VSS and WSS

We now present two different standard definitions of VSS.

Definition 3.1 (Weak Definition of VSS [43]) *In a VSS protocol there is a distinguished party $D \in \mathcal{P}$, that holds an input $s \in \mathbb{F}$ referred to as the secret. The protocol consists of two phases, a sharing phase and a reconstruction phase. We call an n party protocol with adversary $\mathcal{A}_t$ an (n, t) VSS protocol if it satisfies the following conditions for dealer D holding secret s:*

- **Secrecy** *If D is honest then $\mathcal{A}_t$'s view during the sharing phase reveals no information on s.* [2] *More formally, $\mathcal{A}_t$'s view is identically distributed for all different values of s.*
- **Correctness** *If D is honest then the honest parties output s at the end of the reconstruction phase. Moreover, this is true for any choice of the random inputs of the uncorrupted parties and $\mathcal{A}_t$'s randomness.*
- **Strong Commitment** *If D is corrupted, then at the end of the sharing phase there is a value $s^* \in \mathbb{F} \cup \{NULL\}$, such that at the end of reconstruction phase all honest parties output s^*, irrespective of the behavior of the corrupted parties.*

This definition is equivalent to saying that $s^* \in \mathbb{F}$, by fixing a default value in $\mathbb{F}$, which may be output in case the reconstruction ends with a NULL. However, we prefer this form of the definition so as to distinguish it from a stronger definition of VSS [94, 91]. The stronger definition of VSS requires that at the end of the sharing there is a commitment to an actual value in $\mathbb{F}$, i.e. the dealer cannot commit to NULL. Thus, using the above definition points to the fact that NULL is a possible value, instead of setting it to a default value in $\mathbb{F}$.

Protocols that do not satisfy the stronger VSS definition are not suitable for use in multiparty computations. Hence, we also need the stronger definition of VSS [94].

Definition 3.2 (Strong Definition of VSS [94]) *This is same as the previous definition with the following modification in* **Strong Commitment***:*

Strong Commitment. *If D is corrupted, then at the end of the sharing phase there is a value $s^* \in \mathbb{F}$, such that at the end of reconstruction phase all honest parties output s^*, irrespective of the behavior of the corrupted parties.*

[2] If D is corrupted, then s will be known to $\mathcal{A}_t$. In such a case, the secrecy property does not apply.

Definition 3.3 (Weak Definition of Statistical VSS) *A statistical VSS protocol is said to satisfy the weak definition of statistical VSS if the protocol satisfies* **correctness** *and* **strong commitment***, except with error probability ϵ. Moreover, the* **strong commitment** *property is inline with weak definition of VSS given in Definition 3.1. Note that we assume secrecy to be perfect.*

Definition 3.4 (Strong Definition of Statistical VSS) *A statistical VSS protocol is said to satisfy the strong definition of statistical VSS if the protocol satisfies* **correctness** *and* **strong commitment***, except with error probability ϵ. Moreover, the* **strong commitment** *property is inline with strong definition of VSS given in Definition 3.2. Note that we assume secrecy to be perfect.*

The VSS protocol with $n = 3t + 1$ presented in this chapter satisfy the weak definition of statistical VSS, which leave the open question of whether a 2-round VSS protocol can be designed that satisfies the strong definition of statistical VSS. However, when examining the round complexity of VSS as a stand alone application, the weak definition is sufficient. The VSS protocols with $n = 2t + 1$ parties presented in this chapter satisfy strong definition of statistical VSS.

VSS in External Dealer Model: In the external dealer model, the system is assumed to consist of a dealer and n parties. The dealer is considered as an external party. Moreover, the adversary $\mathcal{A}_t$ is allowed to corrupt D and up to t *additional* parties. We stress that all the protocols and lower bounds presented in this chapter will work for this model as well.

We now present the definition of WSS [138, 137].

Definition 3.5 (WSS) *The setting is the same as for the VSS and the definition satisfies the* **Secrecy** *and* **Correctness** *properties. However, we relax the* **Strong Commitment** *property as follows:*

Weak Commitment. *If D is faulty then at the end of the sharing phase there is a value $s^* \in \mathbb{F} \cup \{NULL\}$ such that at the end of the reconstruction phase, each honest party will output either s^* or $NULL$.*

Notice that it is not required that all honest parties output the same value, i.e. some may output s^* and some may output NULL. The above definition is standard and follows many of the existing definitions [137, 138, 109].

Definition 3.6 (Statistical WSS) *A statistical WSS protocol satisfies correctness and weak commitment, except with error probability ϵ. Moreover, secrecy is assumed to be perfect.*

3.1.8 The Road-map

We have structured this chapter in the following way:

1. Section 3.2: Presents 1-round sharing 2-round reconstruction $(4, 1)$ VSS.
2. Section 3.3: Presents 2-round sharing 2-round reconstruction $(3t+1, t)$ WSS.
3. Section 3.4: Presents 2-round sharing 2-round reconstruction $(3t+1, t)$ VSS. This protocol uses 2-round sharing WSS as a black box (presented in section 3.3).

4. Section 3.5: Presents 3-round sharing 2-round reconstruction $(3, 1)$ VSS.
5. Section 3.6: Presents 4-round sharing 2-round reconstruction $(2t+1, t)$ VSS. This protocol is in-efficient.
6. Section 3.7: Presents 5-round sharing 2-round reconstruction *efficient* $(2t+1, t)$ VSS.
7. Section 3.8: Presents the lower bounds on VSS.
8. Section 3.9: Presents 1-round sharing 2-round reconstruction $(3t+1, t)$ WSS.
9. Section 3.10: Presents 3-round sharing 2-round reconstruction $(2t + 1, t)$ WSS.
10. Section 3.11: Presents the lower bounds on WSS.

Finally, this chapter ends with a concluding note where we pose a set of interesting open problems.

3.2 Efficient 1-round Sharing, 2-round Reconstruction $(4, 1)$ Statistical VSS

Here we design a 1-round sharing, 2-round reconstruction $(4, 1)$ statistical VSS protocol. In [91] it is shown that there exists a 1-round sharing, 1-round reconstruction $(5, 1)$ *perfect* VSS. This shows that probabilistically relaxing the conditions of VSS helps to increase the fault tolerance. Let the parties be denoted by P_1, P_2, P_3, P_4, where P_1 is the dealer and s is the secret. The protocol has an error probability of ϵ. To bound the error probability by ϵ, our protocol works over a finite field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using relation $\epsilon \geq 2^{-\kappa}$.

The Intuition: The sharing phase of our VSS protocol is very simple. Here D selects four polynomials each of degree one, namely $f(x), f_2(x), f_3(x), f_4(x)$ such that the constant term of $f(x)$ is secret s and $f(i) = f_i(0)$ for $i = 2, 3, 4$. Moreover D also selects three random non-zero values from $\mathbb{F}$ (called as evaluation points), namely $\alpha_2, \alpha_3, \alpha_4$. D then gives $f_i(x)$, α_i and values of other $f_j(x)$ polynomials at α_i to P_i for $i = 2, \ldots, 4$. Notice that D participates in the sharing phase just to distribute information to other parties. In the reconstruction phase, the parties disclose their polynomials, secret evaluations points and the values of the polynomials as received in sharing phase. The important step is that D is not allowed to take part in reconstruction phase. Now the reconstruction phase is simple and is based on the technique of using secret evaluation points and values (of the polynomials) to check the validity of the polynomials. The protocol is presented in Fig. 3.1

Lemma 3.7 (Secrecy) *Protocol 1-Round-VSS satisfies perfect secrecy.*

Proof: We have to consider the case when D is honest. Without loss of generality, let P_4 be corrupted. Then P_4 knows $f_4(x)$. P_4 will also know one distinct point on each $f_i(x)$ for $2 \leq i \leq 3$. Since degree of each $f_i(x)$ is one, adversary lacks one point on each $f_2(x), f_3(x)$ to completely know them and hence $f(0) = s$ will be information theoretically secure. □

Figure 3.1: 1-Round Sharing, 2-Round Reconstruction $(4, 1)$ Statistical VSS.

Protocol 1-Round-VSS$(D, \mathcal{P}, s, \epsilon)$

Sharing Phase: One Round

1. D selects a random polynomial $f(x)$ over $\mathbb{F}$ of degree one, such that $f(0) = s$.
2. For every i, $2 \leq i \leq 4$ the dealer D chooses and sends to P_i the following:
   (a) A random polynomial $f_i(x)$ of degree one over $\mathbb{F}$ such that $f_i(0) = f(i)$.
   (b) Random non-zero element from $\mathbb{F}$, denoted by α_i.
   (c) $v_{ji} = f_j(\alpha_i)$ for $2 \leq j \leq 4$.

Reconstruction Phase: Two Rounds— $D(P_1)$ is not allowed to participate

Round 1: Each P_i broadcasts $f'_i(x)$, for $2 \leq i \leq 4$.

Round 2: For $2 \leq i \leq 4$, P_i broadcasts the evaluation point α'_i and the values v'_{ji}, for $2 \leq j \leq 4$.

Local Computation (by each party except P_1):

1. Party $P_i \in \mathcal{P} \setminus \{P_1\}$ is *confirmed* if there exists a $P_j \in \mathcal{P} \setminus \{P_1, P_i\}$ for which $f'_i(\alpha'_j) = v'_{ij}$.
2. If there are less then two *confirmed* parties then output $NULL$. Otherwise, if the $f'_i(0)$s corresponding to the set of *confirmed* parties define a polynomial $f(x)$ of degree one then output $f(0)$; else output NULL.

Lemma 3.8 (Correctness) *Protocol 1-Round-VSS satisfies correctness property, except with error probability ϵ.*

PROOF: Here we have to consider D to be honest. If D is honest, then among the remaining three parties at most one can be corrupted. Let P_4 be the corrupted party among P_2, P_3 and P_4. It is easy to see that P_2 and P_3 will be *confirmed*. Therefore there will be at least two *confirmed* parties. Now we assert that if P_4 is *confirmed* then he must have broadcasted $f'_4(x) = f_4(x)$ during reconstruction phase with probability at least $(1-\epsilon)$. So first assume that P_4 broadcasts $f'_4(x) \neq f_4(x)$ during reconstruction phase. Clearly, P_4 will be *confirmed* only if $f'_4(\alpha_2) = f_4(\alpha_2)$ or $f'_4(\alpha_3) = f_4(\alpha_3)$. But since P_4 broadcasts $f'_4(x)$, without knowing α_2 and α_3, the first, second or both the equalities may satisfy only when P_4 can correctly guess α_2, α_3 or both respectively. But P_4 can do the guessing only with probability at most $\frac{2}{|\mathbb{F}|} \approx \epsilon$, which is negligible.

So with probability at least $(1-\epsilon)$, $f'_i(x) = f_i(x)$ for every *confirmed* party. Now it is easy to see that $f(x)$ and hence secret $s = f(0)$ will be reconstructed back with the help of $f_i(0)$ values, except with probability at most ϵ. □

Lemma 3.9 (Strong Commitment) *Protocol 1-Round-VSS satisfies strong commitment property without any error probability.*

Proof: We have to consider the case when D (P_1) is corrupted. Thus P_2, P_3 and P_4 are honest and behave correctly in the reconstruction phase (recall that D is not allowed to participate in the reconstruction phase). As the values of the honest parties are fixed in the sharing phase, the question of which party will be *confirmed* is fixed as well. Thus, D is committed to $NULL$ if (a) there is zero or one *confirmed* party or (b) there are three *confirmed* parties but their $f_i(0)$'s do not define a polynomial $f(x)$ of degree one. On the other hand, we say that D is committed to $f(0)$ when there are (a) two *confirmed* parties whose $f_i(0)$'s define a unique polynomial $f(x)$ of degree one or (b) three *confirmed* parties whose $f_i(0)$'s define a unique polynomial $f(x)$ of degree one. In the reconstruction phase, D's committed secret (which is either NULL or $f(0)$) will be reconstructed without any error. Hence the lemma. □

Theorem 3.10 *There exists an efficient 1-round sharing, 2-round reconstruction* $(4, 1)$ *statistical VSS protocol.*

Proof: Protocol 1-Round-VSS presented here achieves correctness, except with probability ϵ and also satisfies strong commitment and secrecy without any error. This follows from Lemma 3.7, 3.8 and 3.9. Hence the theorem. □

Protocol 1-Round-VSS follows the weak definition of statistical VSS (see Definition 3.3). This follows from the proof of Lemma 3.9 where it is shown that a corrupted D may commit to $NULL$.

3.2.1 Statistical VSS with One Round of Reconstruction

It is interesting to note that if we restrict the adversary to a non-rushing adversary then the two rounds of the reconstruction phase can be collapsed into a single round. The two rounds are needed in order to force the adversary to commit to the polynomial $f_i(x)$ of the faulty party prior to seeing the evaluation points, as this knowledge can enable the adversary to publish a polynomial that will match with the values of the honest parties, which would violate the correctness of the protocol. However, if the adversary is non-rushing then this property is achieved via the synchronicity of the step. We state this in the following theorem:

Theorem 3.11 *If the adversary* $\mathcal{A}_t$ *is non-rushing then there exists an efficient 1-round sharing 1-round reconstruction* $(4, 1)$ *statistical VSS protocol.*

3.2.2 Statistical VSS with No Broadcast

We now show how protocol 1-Round-VSS can be modified, so that it uses no broadcast. The **Sharing Phase** of 1-Round-VSS uses no broadcast (see Fig. 3.1). Now we modify the **Reconstruction Phase**, so that it does not require any broadcast.

Reconstruction Phase: Two Rounds— $D(P_1)$ is not allowed to participate

Round 1: Each P_i privately sends $f'_i(x)$, for $2 \leq i \leq 4$ to every other party P_j.

Round 2: For $2 \leq i \leq 4$, P_i privately sends the evaluation point α'_i and the values v'_{ji}, for $2 \leq j \leq 4$ to every other party P_j.

Local Computation (by each party except P_1): This is same as presented in Fig. 3.1.

This modified version of protocol 1-Round-VSS preserves all the properties of protocol 1-Round-VSS.

3.3 Efficient 2-round Sharing, 2-round Reconstruction $(3t+1, t)$ Statistical WSS

In this section, we present our 2-round sharing, 2-round reconstruction statistical WSS protocol with $n = 3t + 1$. This is used as building block to design our 2-round sharing, 2-round reconstruction $(3t + 1, t)$ statistical VSS presented in the next section. The WSS protocol appears in Fig. 3.2. For ease of exposition, we describe our protocol using multiple rounds of broadcast. We follow this with a brief description on how to modify the protocol to a variation that uses a single round of broadcast. The protocol has an error probability of ϵ. To bound the error probability by ϵ, our protocol works over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^2\kappa 2^{-\kappa}$. So we have $|\mathbb{F}| \geq \frac{n^2\kappa}{\epsilon}$.

The Intuition: In the sharing phase, D selects a bivariate polynomial $F(x, y)$ of degree t in y and $n\kappa + 1$ in x. D also selects κ random secret evaluation points for every party in $\mathcal{P}$. D delivers the polynomial $f_i(x) = F(x, i)$, κ evaluation points and the values of all $f_j(x)$ polynomials (for $j = 1, \ldots, n$) at those κ evaluation points to P_i. In **Round 2**, the parties communicate among themselves to check the consistency of the $f_i(x)$ polynomials with their corresponding values in a zero-knowledge fashion. To keep the secrecy of $f_i(x)$ polynomials during the zero-knowledge communication, D also delivers another random polynomial $r_i(x)$ of degree $n\kappa + 1$ to every P_i and its corresponding values at κ secret evaluations points to every party. The details of the protocol can be found in Fig. 3.2.

Now before we turn to our proofs we draw the readers attention to the following interesting points that enable us to achieve the final result. The bivariate polynomial $F(x, y)$ (defined by D) has a tweak, the x variable is of degree $n\kappa+1$, which results in the polynomials $f_i(x)$ being of degree $n\kappa+1$ (where as this degree is typically t in other protocols). We further create a situation where these polynomials never need to be reconstructed and thus the parties need not hold large number of points on the polynomials to interpolate them. These two properties put together, enable us to give each party many evaluation points and values on these polynomials and to further allow them to expose a portion of them without exposing the underlying polynomial. In addition, we adapt an interesting technique from Tompa and Woll [146] and use *secret* evaluation points.

The fact that we can expose points on the high degree polynomials and that the evaluation points are secret, facilitates the cut-and-choose proof, carried out by the parties in Round 2. It should be noted that if we allow rushing adversary, then a cheating prover may try to foil the cut-and-choose proof during the sharing phase. However, surprisingly we show that this proof is sufficient for our needs and that we can deal with such faulty parties in the reconstruction phase.

NOTE: Following the notation of [91], whenever we say that dealer is disqualified during the sharing phase of WSS/VSS, we mean to say that all honest parties accept the sharing of NULL (or a default value from $\mathbb{F}$) as the dealer's secret. $\square$

Figure 3.2: 2-Round Sharing, 2-Round Reconstruction $(3t+1, t)$ Statistical WSS.

Protocol 2-Round-WSS$(D, \mathcal{P}, s, \epsilon)$

Sharing Phase: Two Rounds

D's Computation: D does the following:

1. Picks a random bivariate polynomial $F(x, y)$ over $\mathbb{F}$ of degree t in the variable y and degree $n\kappa + 1$ in the variable x, such that $F(0, 0) = s$.
2. Defines $f_i(x) = F(x, i)$ for $1 \leq i \leq n$.
3. Picks random polynomials $r_i(x)$ over $\mathbb{F}$, such that $deg(r_i(x)) = n\kappa + 1$ for $1 \leq i \leq n$.
4. Chooses $n\kappa$ random, non-zero, distinct elements from $\mathbb{F}$, denoted by $\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,\kappa}$ for $1 \leq i \leq n$.

Round 1: D sends to party P_i:

- The polynomials $f_i(x), r_i(x)$. Let $f_i(0)$ be P_i's share of D's secret s.
- The random evaluation points $\alpha_{i,\ell}$ for $1 \leq \ell \leq \kappa$.
- $a_{j,i,\ell} = f_j(\alpha_{i,\ell})$ and $b_{j,i,\ell} = r_j(\alpha_{i,\ell})$ for $1 \leq \ell \leq \kappa, \ \ 1 \leq j \leq n$.

Round 2: Party P_i broadcasts the following:

- A random non-zero value c_i and polynomial $g_i(x) = f_i(x) + c_i r_i(x)$ with $deg(g_i(x)) = n\kappa + 1$. (*Whenever we say that a party broadcasts a polynomial of a certain degree we assume that if this is not done then the party is disqualified.*)
- For a random subset of indices $\ell_1, ..., \ell_{\frac{\kappa}{2}}$, the evaluation points $\alpha_{i,\ell_1}, ..., \alpha_{i,\ell_{\frac{\kappa}{2}}}$ and $a_{j,i,\ell_1}, ..., a_{j,i,\ell_{\frac{\kappa}{2}}}$ and $b_{j,i,\ell_1}, ..., b_{j,i,\ell_{\frac{\kappa}{2}}}$ for $1 \leq j \leq n$.

Local Computation (By Every Party):

1. Party P_i is *accepted* by party P_j if $a_{i,j,\ell} + c_i b_{i,j,\ell} = g_i(\alpha_{j,\ell})$ for all ℓ in the set of indices broadcasted by P_j in **Round 2**.
2. Initiate the set $SH = \emptyset$. Place P_i in SH if it is accepted by at least $2t + 1$ parties.
3. If $|SH| \leq 2t$ disqualify dealer D. Note that SH computed by all honest parties are identical.

Reconstruction Phase: Two Rounds

Round 1: Each P_i in SH broadcasts $f_i(x)$ such that $deg(f_i(x)) = n\kappa + 1$.

Round 2: Each $P_j \in \mathcal{P}$ broadcasts all the evaluation points $\alpha_{j,\ell}$ which were not broadcasted in the sharing phase and $a_{i,j,\ell}$ corresponding to those indices, for $i = 1, \ldots, n$.

Local Computation (By Every Party):

1. Party $P_i \in SH$ is *re-accepted* by $P_j \in \mathcal{P}$ if for one of the newly revealed points it holds that $a_{i,j,\ell} = f_i(\alpha_{j,\ell})$.
2. Initiate the set $REC = \emptyset$. Place P_i in REC if it is *re-accepted* by at least $t + 1$ parties.
3. If $|REC| < t + 1$, then output $NULL$. Otherwise, if the shares (i.e $f_i(0)$'s) of the parties in REC interpolate to a degree t polynomial $g(y)$ then output $s = g(0)$; else output $NULL$.

Lemma 3.12 (Secrecy) *Protocol 2-Round-WSS satisfies perfect secrecy.*

PROOF: The secrecy has to be argued when D is honest. For simplicity, assume that first t parties are corrupted. So in **Round 1** of the **Sharing Phase**, the adversary will know the polynomials $f_1(x), \ldots, f_t(x), r_1(x), \ldots, r_t(x)$ and κt points on $f_i(x)$ and $r_i(x)$ for $t+1 \leq i \leq n$. In **Round 2** of the **Sharing Phase**, the adversary learns $\frac{\kappa}{2}(2t+1)$ additional points on $f_i(x)$ and $r_i(x)$ for $t+1 \leq i \leq n$. So in total the adversary will know $\kappa t + \frac{\kappa}{2}(2t+1)$ points on each of $f_i(x)$ and $r_i(x)$ for $t+1 \leq i \leq n$ which is less than the degree of the polynomials, i.e $(n\kappa + 1)$. Thus, the constant term of the polynomials $f_i(x)$ for $t+1 \leq i \leq n$ are information theoretically secure in the **Sharing Phase**, which further implies information theoretic security for s. □

Lemma 3.13 (Correctness) *Protocol 2-Round-WSS satisfies correctness property, except with probability ϵ.*

PROOF: It is easy to see that if D is honest, then every honest party P_i is present in SH as well as in REC. Given that all honest parties are present in SH, the dealer will not be disqualified during the sharing phase. In order to show that the correct secret is reconstructed, we first prove that if a faulty P_i (belonging to SH) broadcasts a polynomial $\overline{f_i}(x) \neq f_i(x)$, then with probability at least $(1 - \frac{\epsilon}{n})$, P_i will not be added to REC.

In order for a faulty P_i to be included in REC, it needs to be *re-accepted* by $t+1$ parties and thus by at least one honest party. For this, the faulty P_i have to guess one of the $\frac{\kappa}{2}$ un-revealed random evaluation points held by some honest party in $\mathcal{P}$. The corrupted P_i can guess one of the $\frac{\kappa}{2}$ un-revealed points for a particular honest P_j with probability at most $\frac{\kappa/2}{|\mathbb{F}|}$. Therefore, P_i can guess one of the $\frac{\kappa}{2}$ un-revealed points for some honest party in $\mathcal{P}$ with probability at most $\mathcal{O}(n)\frac{\kappa/2}{|\mathbb{F}|} \approx \frac{(n\kappa)}{|\mathbb{F}|} \leq \frac{\epsilon}{n}$. Thus we have proved that if a faulty P_i (belonging to SH) broadcasts a polynomial $\overline{f_i}(x) \neq f_i(x)$ in reconstruction phase, then with probability at least $(1 - \frac{\epsilon}{n})$, P_i will not be added to REC. Subsequently, none of the faulty parties of SH who broadcast a polynomial $\overline{f_i}(x) \neq f_i(x)$ will be included in REC with probability at least $(1 - \mathcal{O}(t)\frac{\epsilon}{n}) \approx (1 - \epsilon)$ (since we may have $\mathcal{O}(t)$ such faulty parties in SH).

The above argument proves that with probability at least $(1 - \epsilon)$, every party in REC have broadcasted the polynomial that he has received from D in sharing phase. Hence with probability at least $(1 - \epsilon)$, the parties will reconstruct $s = f(0)$, which is D's secret. □

Note that in the previous proof we did not claim, and in fact cannot claim, that there are no faulty parties in SH. As we allow the adversary to be rushing, it can cause faulty parties, i.e. parties that have broadcasted inconsistent polynomials (during **Round 2** of the sharing phase), to be included in this set. This is done by waiting to hear the evaluation points of the honest parties (in the **Round 2** of the sharing phase). However, this does not affect the result of the reconstruction because the parties in SH broadcast their polynomials in the **Round 1** while the secret evaluation points of the parties are revealed only in the **Round 2** of the reconstruction phase.

Lemma 3.14 (Weak Commitment) *Protocol 2-Round-WSS satisfies weak commitment property, except with probability at most ϵ.*

PROOF: To prove this lemma we need to show that in case a faulty D was not disqualified, i.e. $|SH| \geq 2t+1$, then with probability at least $(1 - \epsilon)$, all the honest parties that are in SH are also present in REC. If we prove this then the lemma follows immediately; we set D's committed secret s^* to be the constant term of the polynomial, which is defined by the interpolation of the shares of the honest parties in SH (note that s^* may be $NULL$). As we require that the shares of all the parties in REC define a polynomial of degree t, then either the value s^* or $NULL$ will be reconstructed.

In order for an honest P_i to be in SH and not in REC it must be the case that at least $2t+1$ parties have *accepted* P_i in the sharing phase but at most t of them *re-accepted* it in the reconstruction phase. This means that there is at least one honest P_j who *accepted* P_i but did not *re-accept* it. This implies that the data (evaluation points and values) that P_j exposed in the sharing phase satisfies the polynomial $g_i(x)$ that P_i broadcasted during the sharing phase, but on the other hand, out of the remaining evaluation points that are used by P_j in the reconstruction phase, none satisfy the polynomial $f_i(x)$ produced by P_i. That is, for the selected $\frac{\kappa}{2}$ indices $\ell_1, ..., \ell_{\frac{\kappa}{2}}$, it holds that $a_{i,j,\ell} + c_i b_{i,j,\ell} = g_i(\alpha_{j,\ell})$ for all ℓ in the set of indices $\{\ell_1, ..., \ell_{\frac{\kappa}{2}}\}$ and $f_i(\alpha_{j,\ell}) \neq a_{i,j,\ell}$ for all ℓ in the *remaining* set of indices. Notice that P_i chooses c_i independent of the values given by D. Also, P_j chooses the $\frac{\kappa}{2}$ indices randomly out of κ indices. So the probability that the above event happens is $\frac{1}{\binom{\kappa}{\kappa/2}} < \frac{1}{2^\kappa} \leq \frac{\epsilon}{n^2\kappa}$. Now the probability that P_i was *accepted* by $2t+1$ parties (in which at least $t+1$ were honest) and is not *re-accepted* by some honest P_j is at most $\mathcal{O}(t)\frac{\epsilon}{n^2\kappa} \approx \frac{\epsilon}{n\kappa}$. Subsequently, we can assert that the above may happen for some honest P_i in SH (i.e some P_i in SH may not belong to REC) with probability at most $\mathcal{O}(t)\frac{\epsilon}{n\kappa} \approx \frac{\epsilon}{\kappa} < \epsilon$.

This shows that all honest parties from SH will be included in REC, with probability exceeding $(1-\epsilon)$. Now consider the case when s^*, the secret defined by the shares of the parties in SH, is a value from $\mathbb{F}$. In this case, depending on how the corrupted parties in REC have exposed their polynomials, either s^* or NULL will be reconstructed. On the other hand if $s^* =$ NULL, then irrespective of the polynomials broadcasted by the corrupted parties in REC, NULL will be reconstructed. □

Theorem 3.15 *There exists an efficient 2-round sharing, 2-round reconstruction $(3t+1, t)$ statistical WSS protocol.*

PROOF: Protocol 2-Round-WSS presented here achieves correctness and weak commitment except with probability ϵ and also achieves perfect secrecy. This follows from Lemma 3.12, 3.13 and 3.14. □

IMPORTANT NOTE: There is another interesting way to interpret the computation done in the protocol 2-Round-WSS. We may view this as D sharing a degree t polynomial $g(y)$ using protocol 2-Round-WSS. For this, D selects the bivariate polynomial $F(x, y)$ as in protocol 2-Round-WSS, such that $F(0, y) = g(y)$. The polynomial $g(y)$ is the polynomial that D used to share the secret $g(0) = F(0, 0) = s$. The polynomial $g(y)$ is not random but only preserves the secrecy of the constant term. Yet, this distribution of polynomials is sufficient to provide the secrecy requirements needed by our protocols.

In the sequel, we will invoke our WSS as 2-Round-WSS$(D, \mathcal{P}, g(y), \epsilon)$ to mean that D want to share $g(y)$ in a sense described above.

3.3.1 Statistical WSS with One Round of Reconstruction

It is interesting to note that if we restrict the adversary to a non-rushing adversary then the two rounds of the reconstruction phase can be collapsed into a single round. The two rounds are needed in order to force the adversary to commit to the polynomials $f_i(x)$ of the faulty parties prior to seeing the evaluation points, as this knowledge can enable the adversary to publish a polynomial that is *re-accepted* by the honest parties, which would violate the correctness of the protocol. However, if the adversary is non-rushing then this property is achieved via the synchronicity of the step. We state this in the following theorem:

Theorem 3.16 *If the adversary is non-rushing then there exists an efficient 2-round sharing, 1-round reconstruction* $(3t + 1, t)$ *statistical WSS protocol.*

3.3.2 Statistical WSS with One Round of Broadcast

We now show how protocol 2-Round-WSS can be modified, so that it uses *only one* round of broadcast (the **Round 2** of **Sharing Phase**). Specifically, we modify the **Reconstruction Phase** of 2-Round-WSS, so that it requires no broadcast.

Reconstruction Phase, 2-rounds:

Round 1: Each P_i in SH privately sends $f_i(x)$, $deg(f_i(x)) = n\kappa + 1$ to every other party.

Round 2: Each $P_j \in \mathcal{P}$ privately sends all the evaluation points $\alpha_{j,\ell}$ which were not broadcasted in the sharing phase and $a_{i,j,\ell}$ for those indices, to all other parties.

Local Computation (By Every Party): It is the same as in the protocol 2-Round-WSS.

This modified version of 2-Round-WSS preserves secrecy perfectly and correctness except with probability at most ϵ. It will also satisfy weak commitment (except with probability at most ϵ), but *without agreement*. That is, some honest party(ies) may output the committed secret s^* while some other may output $NULL$.

3.4 Efficient 2-round Sharing, 2-round Reconstruction $(3t+1, t)$ Statistical VSS

We now design a 2-round sharing, 2-round reconstruction $(3t + 1, t)$ statistical VSS protocol. In [91] it is shown that there exists a 2-round sharing, 1-round reconstruction $(4t + 1, t)$ *perfect* VSS. This shows that probabilistically relaxing the conditions of VSS helps to increase the fault tolerance.

The Intuition: We follow the general idea of [20, 91, 73, 109] of sharing the secret s with a symmetric bivariate polynomial $F(x, y)$ where each party P_i gets the univariate polynomial $f_i(y) = F(i, y)$ and his share is $f_i(0)$. The next step is for every pair of parties to verify that they have received the correct values from the dealer. However, as we have only one more round available we cannot depend on D to resolve conflicts in a third round. Thus, instead of doing the verification point wise we carry out the verification on polynomials. More specifically, party P_i initiates an execution of protocol 2-Round-WSS in the first round, to share a random polynomial $g_i(y)$. In the second round, P_i broadcasts the masked polynomial $h_i(y) = f_i(y) + g_i(y)$, while every other party broadcasts the corresponding point on $h_i(y)$. In fact, this verification can be viewed as an extension of the round reducing technique of pad sharing for a single value given in [91], to the sharing of polynomial, which is used as a pad for the verification of a polynomial. Our 2-round sharing VSS protocol now appears in Fig. 3.3.

The protocol has a error probability of ϵ. To bound the error probability by ϵ, our protocol works over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^3\kappa 2^{-\kappa}$. This is because, our protocol invokes 2-Round-WSS with error probability $\frac{\epsilon}{n}$ and we require $\epsilon \geq n^2\kappa 2^{-\kappa}$ to bound the error probability of 2-Round-WSS by ϵ. Now the relation $\epsilon \geq n^3\kappa 2^{-\kappa}$ implies that we have $|\mathbb{F}| \geq \frac{n^3\kappa}{\epsilon}$.

Lemma 3.17 (Secrecy) *Protocol 2-Round-VSS satisfies perfect secrecy.*

Proof: This proof is similar to the entropy based argument, used to prove the secrecy of 3-round perfect VSS protocol of [73]. □

Lemma 3.18 (Correctness) *Protocol 2-Round-VSS satisfies correctness property, except with probability ϵ.*

Proof: A simple examination of the protocol 2-Round-VSS and the properties of protocol 2-Round-WSS reveal that all honest parties will be in VSS-SH and thus an honest D is not disqualified during the sharing phase. Now to prove this lemma we need to show two things:

- With probability at least $(1-\epsilon)$, for every faulty party P_j in REC the following holds: if at the end reconstruction phase of WSS^{P_j}, the reconstructed polynomial is $g_j(y)$, then $h_j(y) - g_j(y)$ is in fact polynomial $f_j(y)$, received by P_j from D. What this implies is that we cannot guarantee that all parties in VSS-SH are honest. But we can ensure that if they eventually remain in REC then they have shared the proper polynomials.
- With probability at least $(1-\epsilon)$, all $f_i(y)$ polynomials corresponding to the honest parties in VSS-SH will be reconstructed successfully (due to the correctness of 2-Round-WSS) and thus there will be sufficient number of parties in REC even when the reconstruction phase of WSS^{P_j} outputs NULL for every corrupted party P_j in VSS-SH.

If we prove the above statements then the lemma follows immediately.

We now prove the first statement. Let P_j be a corrupted party in REC. Evidently, P_j belongs to VSS-SH. Now since P_j is present in VSS-SH, we know that $|Accept_j \cap SH_j| \geq 2t+1$. This means that there are $t+1$ honest parties in this set. By the properties of 2-Round-WSS, this set of honest parties define the polynomial $g_j(y)$ which P_j is committed to, at the end of the sharing phase of WSS^{P_j}. We now examine the polynomial $h_j(y) - g_j(y)$ and show that it is equal to $f_j(y)$. The set of $(t+1)$ honest parties in $(Accept_j \cap SH_j)$ verified that the sum of the share $f_i(j) = f_j(i)$ (which they received from D) and $g_j(i)$ (which they received from P_j), in fact lie on the polynomial $h_j(y)$. Moreover, the set of

Figure 3.3: 2-Round Sharing, 2-Round Reconstruction $(3t+1, t)$ Statistical VSS.

Protocol 2-Round-VSS$(D, \mathcal{P}, s, \epsilon)$

Sharing Phase: Two Rounds

Round 1:

- D selects a random symmetric bivariate polynomial $F(x, y)$ over $\mathbb{F}$ of degree t in each variable such that $F(0, 0) = s$ and sends the polynomial $f_i(y) = F(i, y)$ to P_i.
- Party P_i initiates **Round 1** of protocol 2-Round-WSS$(P_i, \mathcal{P}, g_i(y), \frac{\epsilon}{n})$ to share a random degree t polynomial $g_i(y)$. Denote this execution by WSS^{P_i}.

Round 2:

- Party P_i broadcasts the polynomial $h_i(y) = f_i(y) + g_i(y)$ such that $deg(h_i(y)) = t$ and values $a_{ji} = f_i(j) + g_j(i) = f_j(i) + g_j(i)$, for $1 \leq j \leq n$.
- Execute **Round 2** of the sharing phase of each WSS^{P_i}. Let SH_i denote the set SH from this execution.

Local Computation (By Every Party):

1. Party P_i *is accepted* by party P_j if $h_i(j) = a_{ij}$.
2. Let $Accept_i$ denote the set of parties that accepted P_i.
3. Create the set VSS-SH. Place P_i in VSS-SH if $|Accept_i| \geq 2t + 1$.
4. Remove P_i from VSS-SH if $|\text{VSS-}SH \cap Accept_i \cap SH_i| \leq 2t$. Repeat, until no more parties can be removed.
5. If $|\text{VSS-}SH| \leq 2t$ then disqualify D.

Reconstruction Phase: Two Rounds

Round 1 and 2: For all P_i in VSS-SH, execute the 2-round reconstruction phase of WSS^{P_i}. If the output of the execution is not NULL then let $g_i(y)$ be the output from this execution.

Local Computation (for each party)

1. Initialize $REC = \text{VSS-}SH$.
2. Remove P_i from REC if the output of WSS^{P_i} is $NULL$.
3. If $|REC| < t + 1$, then output NULL. Otherwise, compute $f_i(y) = h_i(y) - g_i(y)$ for all $P_i \in REC$ such that $g_i(y)$ is obtained from the reconstruction phase of WSS^{P_i} and $h_i(y)$ was broadcasted by P_i in **Round 2** of sharing phase.
4. For each $(P_i, P_j) \in REC$, check whether $f_i(j) = f_j(i)$. If not then output NULL. If yes, then reconstruct symmetric bivariate polynomial $F(x, y)$ such that $F(x, i) = f_i(x)$ for every $P_i \in REC$ and output $F(0, 0)$.

$t + 1$ shares, corresponding to these honest parties define the polynomial $f_j(y)$.

Thus, $h_j(y) - g_j(y) = f_j(y)$. Now by the weak commitment property of protocol 2-Round-WSS, $g_j(y)$ has been reconstructed correctly, with probability $(1 - \frac{\epsilon}{n})$. Since REC may contain at most t corrupted parties, the probability that $g_j(y)$ corresponding to all of them will be reconstructed correctly, is $(1 - t\frac{\epsilon}{n}) \approx (1 - \epsilon)$. Thus in the reconstruction phase of our VSS protocol, $h_j(y) - g_j(y)$ will be polynomial $f_j(y)$, received by P_j from D for all corrupted P_j in REC, with probability at least $(1 - \epsilon)$.

We now prove the second statement. The reconstruction phase of 2-Round-WSS corresponding to an honest party in VSS-SH will be successful with probability $(1 - \frac{\epsilon}{n})$ (according to the correctness property of 2-Round-WSS). Now since there are $2t+1$ honest parties in VSS-SH, the probability that the reconstruction phase of 2-Round-WSS corresponding to all the honest parties in VSS-SH will be successful is at least $(1 - (2t + 1)\frac{\epsilon}{n}) \approx (1 - \epsilon)$.

Now it is easy to see that for an honest D, the secret $s = F(0, 0)$ will be reconstructed correctly, except with probability ϵ. □

Lemma 3.19 (Strong Commitment) *Protocol 2-Round-VSS satisfies strong commitment property, except with probability ϵ.*

Proof: If D is corrupted and does not get disqualified during the sharing phase, then VSS-SH is fixed at the end of sharing phase. Since VSS-$SH \geq 2t + 1$, it contains a set $\mathcal{H}$ of honest parties of size at least $t + 1$. If $f_j(y)$'s corresponding to the parties in $\mathcal{H}$ define a unique symmetric bivariate polynomial $F^*(x, y)$ of degree t in x and y, then D's committed secret is $s^* = F^*(0, 0)$. Otherwise, $s^* =$ NULL. We show that in the reconstruction phase s^* will be reconstructed, with probability at least $(1 - \epsilon)$.

It is easy to see that due to the correctness property of our 2-Round-WSS, with probability at least $(1 - \epsilon)$, all the honest parties in $\mathcal{H} \subseteq$ VSS-SH will also be present in REC. We now divide our proof into two cases: (a) $s^* \neq$ NULL: The proof for this case follows from the proof of Lemma 3.18 as this case is indistinguishable from the case when D is honest. (b) $s^* =$ NULL: As $\mathcal{H} \subseteq REC$, during Step 3 of the reconstruction phase all parties will output NULL which is equal to s^* with probability at least $(1 - \epsilon)$. Hence the lemma. □

Theorem 3.20 *There exists an efficient 2-round sharing, 2-round reconstruction $(3t + 1, t)$ statistical VSS protocol.*

Proof: Protocol 2-Round-VSS presented here achieves correctness and strong commitment except with probability ϵ and also satisfies perfect secrecy. This follows from Lemma 3.17, 3.18 and 3.19. □

We stress that protocol 2-Round-VSS follows weak definition of statistical VSS as presented in Definition 3.3. That is, in 2-Round-VSS, D can commit $NULL$ at the end of the sharing phase. This makes Protocol 2-Round-VSS unsuitable for Multiparty Computation. It is an interesting problem to see whether there exists an efficient 2-round sharing, $(3t+1, t)$ statistical VSS protocol, which satisfies the strong definition of statistical VSS [94, 91], (see Definition 3.4 given in Section 3.1). In fact, if such a VSS exists then it would also imply that there is a one round reconstruction, as error correction can be used to interpolate the secret in the reconstruction phase.

3.4.1 Statistical VSS with One Round of Reconstruction

As the reconstruction phase of the 2-Round-VSS is simply the reconstruction phase of 2-Round-WSS, we claim here as well, that the reconstruction phase can be collapsed into one round against a non-rushing adversary.

Theorem 3.21 *If the adversary is non-rushing then there exists an efficient 2-round sharing 1-round reconstruction $(3t+1, t)$ statistical VSS protocol.*

3.4.2 Statistical VSS with One Round of Broadcast

We now explain how protocol 2-Round-VSS can be modified, so that the broadcast channel is used in only one round throughout the protocol, namely in **Round 2** of the sharing phase. The reconstruction phase of 2-Round-VSS is simply the reconstruction phase of 2-Round-WSS. Moreover, in the previous section, we have seen how protocol 2-Round-WSS can be modified, so as to have only one round of broadcast. Thus, if we can argue that the modified 2-Round-WSS is sufficient for the reconstruction of 2-Round-VSS, then we have a VSS protocol that does not use broadcast in the reconstruction phase. Examining the proof of 2-Round-VSS, we see that it is not mandatory that the set of polynomials, which the honest parties use in reconstruction is identical, but rather that it has a large enough intersection. As the polynomials of the honest parties provide this guarantee, it is irrelevant which polynomials of the faulty parties are included in the computation. Thus, by using the modified 2-Round-WSS, we get a 2-round sharing, 2-round reconstruction statistical VSS, with only one round of broadcast.

3.5 Efficient 3-round Sharing, 2-round Reconstruction $(3, 1)$ Statistical VSS

We now present a 3-round sharing, 2-round reconstruction $(3, 1)$ statistical VSS protocol called 3-Round-VSS. As opposed to the previous VSS protocols, here the protocol ensures that D always selects secret from $\mathbb{F}$ instead of $\mathbb{F} \cup NULL$. That is protocol 3-Round-VSS satisfies the strong definition of statistical VSS (see Definition 3.4). Let the three parties be denoted as $\mathcal{P} = \{P_1, P_2, P_3\}$ with $D = P_1$. The protocol is now given in Fig. 3.4.

The protocol has an error probability ϵ. To bound the error probability by ϵ, our protocol 3-Round-VSS operates over field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq 2n2^{-\kappa}$. Hence $|\mathbb{F}| \geq \frac{2n}{\epsilon}$. This is because, our protocol invokes MVMS-ICP with error parameter $\frac{\epsilon}{2}$ and $\epsilon \geq n2^{-\kappa}$ should hold to bound the error probability of MVMS-ICP by ϵ.

Lemma 3.22 (Secrecy) *Protocol 3-Round-VSS satisfies secrecy property.*

PROOF: Easy. Follows from Lemma 2.6 and the fact that $f(x)$ is a degree one polynomial and the adversary $\mathcal{A}_t$ has only one point on it. □

Lemma 3.23 (Correctness) *Protocol 3-Round-VSS satisfies correctness property, except with probability ϵ.*

PROOF: We have to consider the case when D is honest. Since D $(= P_1)$ is honest, either P_2 or P_3 is corrupted. Without loss of generality, let P_2 be the corrupted

Figure 3.4: A 3-Round Sharing 2-Round Reconstruction $(3, 1)$ Statistical VSS protocol.

3-Round-VSS($D, \{P_1, P_2, P_3\}, s, \epsilon$)

Sharing Phase: Three Rounds

Round 1:

1. D chooses a random degree one polynomial $f(x)$ such that $f(0) = s$.
2. For $i = 1, 2, 3$, D passes $ICSig(D, P_i, \mathcal{P}, s_i)$ having $\frac{\epsilon}{2}$ error to party P_i, where $s_i = f(i)$.

Round 3: Every P_i receives $ICSig(D, P_i, \mathcal{P}, s_i)$ having $\frac{\epsilon}{2}$ error.

Reconstruction Phase: Two Rounds

Round 1 and 2: For $i = 1, \ldots, 3$, party P_i reveals $ICSig(D, P_i, \mathcal{P}, s_i)$ having $\frac{\epsilon}{2}$ error.

Local Computation (By Each Party)

1. If P_2 and P_3 are successful in revealing $ICSig(D, P_2, \mathcal{P}, s_2)$ and $ICSig(D, P_3, \mathcal{P}, s_3)$ respectively, then assign $REC = \{P_2, P_3\}$. Otherwise, let REC be the set of P_i's who are successful in revealing $ICSig(D, P_i, \mathcal{P}, s_i)$.
2. Let $f(x)$ be the degree one polynomial defined by the s_i values corresponding to the parties in REC. Output $s = f(0)$ as the secret and terminate.

party. Thus P_3 will always be present in REC. From **IC-Correctness3**, every $ICSig(D, P_2, \mathcal{P}, s'_2)$ with $s'_2 \neq s_2$ revealed by P_2 will be rejected, except with probability $\frac{\epsilon}{2}$ and P_2 will not be included in REC. So it is assured that if P_2 is in REC then $s_2 = f(2)$, with probability $(1 - \frac{\epsilon}{2})$. Now depending on the behavior of P_2, REC will contain either $\{P_2, P_3\}$ or $\{P_1, P_3\}$. In both the cases, $f(x)$ will be interpolated back and $s = f(0)$ will be taken as the secret, with probability at least $(1 - \frac{\epsilon}{2})$. We can prove the above when P_3 instead of P_2 is corrupted. Therefore our protocol satisfies correctness property, with probability at least $(1 - 2\frac{\epsilon}{2}) = (1 - \epsilon)$. Hence the lemma. □

Lemma 3.24 (Strong Commitment) *Protocol 3-Round-VSS satisfies strong commitment property, except with probability ϵ.*

Proof: We have to consider the case when D is corrupted. Notice that as $t = 1$ and D is corrupted, both P_2 and P_3 are honest here. So we define D's committed secret $\overline{s}$ as the constant term of the degree one polynomial say $\overline{f}(x)$ defined by the s_2 and s_3 held by P_2 and P_3 respectively. Now by **ICP-Correctness2**, both P_2 and P_3 will be successful in revealing $ICSig(D, P_2, \mathcal{P}, s_2)$ and $ICSig(D, P_3, \mathcal{P}, s_3)$ respectively, except with probability $2\frac{\epsilon}{2} = \epsilon$. Thus REC will contain only P_2 and P_3 and $\overline{s}$ will be reconstructed with probability $(1 - \epsilon)$. □

Theorem 3.25 *There exists an efficient 3-round sharing, 2-round reconstruction $(3, 1)$ statistical VSS protocol.*

PROOF: Protocol 3-Round-VSS presented here achieves correctness and strong commitment, except with error probability ϵ and also achieves perfect secrecy. This follows from Lemma 3.22, 3.23 and 3.24. □

It is to be noted that the number of rounds in reconstruction phase of 3-Round-VSS is optimal from the results of [49] which proves the necessity (and sufficiency) of two rounds in reconstruction phase for any VSS with $n = 2t + 1$ and $t \geq 1$. But we can have a 3-round sharing 1-round reconstruction VSS if we consider the adversary to be non-rushing (as shown in the next section).

3.5.1 3-round Sharing VSS with One Round of Reconstruction

If we restrict the adversary to a non-rushing adversary then the two rounds of reconstruction phase of protocol 3-Round-VSS can be collapsed into a single round. This is because the reconstruction phase of protocol 3-Round-VSS is nothing but the execution of Reveal of MVMS-ICP which can be achieved in single round when adversary is non-rushing. Hence, we have the following theorem:

Theorem 3.26 *If the adversary is non-rushing then there exists an efficient 3-round sharing, 1-round reconstruction* $(3, 1)$ *statistical VSS protocol.*

3.6 In-efficient 4-round Sharing, 2-round Reconstruction $(2t + 1, t)$ Statistical VSS

In this section, we present a 4-round sharing 2-round reconstruction $(2t + 1, t)$ statistical VSS protocol that takes exponential communication and computation complexity. In [91] it is shown that there exists a 4-round sharing, 1-round reconstruction $(3t+1, t)$ *perfect* VSS. This shows that probabilistically relaxing the conditions of VSS helps to increase the fault tolerance. Here the protocol ensures that D always selects secret from $\mathbb{F}$ instead of $\mathbb{F} \cup NULL$. That is, our protocol satisfies the strong definition of statistical VSS (see Definition 3.4).

The Intuition: Let $S_1, \ldots, S_K$ be an enumeration of all $K = \binom{n}{t+1}$ subsets of $n - t = t + 1$ parties. In the sharing phase of our protocol (called 4-Round-VSS), D additively shares the secret s into $s_1, \ldots, s_K$ where $s_1, \ldots, s_K$ are random, subject to $s = s_1 + s_2 + \ldots + s_K$. D delivers s_i to all the $t + 1$ parties in set S_i. Next, the parties in S_i communicate among themselves to check whether they all received the same value from D. If there is any confliction between any pair of parties in a set S_i, then D is sure that at least one party in the pair is faulty and therefore broadcasts s_i. Notice that when D is honest, this does not violate secrecy property of VSS as there will be at least one set S_i that contains all the $t + 1$ honest parties and they will never conflict with each other (and therefore D will never broadcast the share s_i for the set S_i).

In the reconstruction phase, for all the sets for which D did not broadcast the value s_i, D's committed value for the set s_i will be reconstructed correctly irrespective of whether D is honest or faulty. To ensure the above properties, our protocol 4-Round-VSS uses IC signatures. The protocol is now given in Fig. 3.5 and 3.6.

Our protocol has an error probability of ϵ. To bound the error probability by ϵ, the computation in our statistical VSS protocol is performed over a field

$\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^3 K 2^{-\kappa}$. This is derived from the fact that in our VSS protocol, MVMS-ICP will be invoked with $\frac{\epsilon}{n^2 K}$ error probability and as mentioned in Chapter 2, $\epsilon \geq n 2^{-\kappa}$ should hold to bound error probability of MVMS-ICP by ϵ.

Figure 3.5: Sharing Phase of 4-round sharing 2-round reconstruction $(2t+1, t)$ statistical VSS.

4-Round-VSS$(D, \mathcal{P}, s, \epsilon)$

Sharing Phase: Four Rounds

Round 1:

1. D additively shares s into $s_1, \ldots, s_K$ where $s_1, \ldots, s_K$ are random subject to $s = s_1 + s_2 + \ldots + s_K$. Then D gives s_k to every party P_i in the subset S_k (which contains $(t+1)$ parties).
2. For each pair (P_i, P_j) from subset S_k for $k = 1, \ldots, K$, party P_i picks a random value $r_{ij}^k \in \mathbb{F}$ and sends $ICSig(P_i, P_j, \mathcal{P}, r_{ij}^k)$ having $\frac{\epsilon}{n^2 K}$ error to P_j. *The r_{ij}^ks will be used by P_i and P_j to check the equality of their common share s_k handed over by D in the subsequent rounds.*

Round 2:

1. Every party $P_i \in S_k$ broadcasts: (a) $a_{ij}^k = s_k + r_{ij}^k$ and (b) $b_{ij}^k = s_k + r_{ji}^k$ for every party $P_j \in S_k$.

Round 3:

1. For every pair of parties (P_i, P_j) from subset S_k for $k = 1, \ldots, K$, P_j receives $ICSig(P_i, P_j, \mathcal{P}, r_{ij}^k)$ having $\frac{\epsilon}{n^2 K}$ error from P_i. An *ordered* pair (P_i, P_j) is called as **conflicting pair** if one of the following holds:
   - Corresponding to some subset S_k, party P_i has broadcasted r_{ij}^k in the **Round 2** of $\mathsf{Ver}(P_i, P_j, \mathcal{P}, r_{ij}^k, \frac{\epsilon}{n^2 K})$.
   - If $a_{ij}^k \neq b_{ji}^k$ for some subset S_k.

Round 4:

1. For every **conflicting** pair (P_i, P_j), D broadcasts s_k, the share for subset S_k for all S_k containing the pair (P_i, P_j).
2. Let $\mathcal{B} = \{S_k \mid D \text{ has broadcasted } s_k\}$. $\overline{\mathcal{B}}$ contains the remaining subsets. If $\overline{\mathcal{B}} = \emptyset$, then discard D.

Lemma 3.27 (Secrecy) *Protocol 4-Round-VSS satisfies perfect secrecy.*

PROOF: We have to consider the case when D is honest. Since the number of parties is $2t+1$, there is one particular subset S_k containing all the $(t+1)$ honest parties. D will never broadcast the share s_k corresponding to S_k as a pair of honest parties will never be a **conflicting pair**. So the corrupted parties will not know the share s_k corresponding to subset S_k and hence s will be information theoretically secure during the **Sharing Phase**. $\square$

Figure 3.6: Reconstruction Phase of 4-round sharing 2-round reconstruction $(2t+1, t)$ statistical VSS.

4-Round-VSS$(D, \mathcal{P}, s, \epsilon)$

Reconstruction Phase: Two Rounds

Round 1 and 2: For every pair of parties (P_i, P_j) from subset $S_k \in \overline{\mathcal{B}}$, P_j reveals $ICSig(P_i, P_j, \mathcal{P}, r_{ij}^k)$ having $\frac{\epsilon}{n^2 K}$ error.

Local Computation (By Each Party):

1. Let s_k be D**'s commitment to subset** S_k where:
   (a) If $S_k \in \mathcal{B}$: then s_k is the one broadcasted by D during **Round 4** of **Sharing Phase**.
   (b) If $S_k \in \overline{\mathcal{B}}$: then s_k is computed as follows:
      i. Let $GOOD_k$ be set of all P_i's in S_k such that: (a) P_i is successful in revealing $ICSig(P_j, P_i, \mathcal{P}, r_{ji}^k)$ for every $P_j \in S_k$. (b) s_{ji}^k values for all $P_j \in S_k$ are equal where $s_{ji}^k = b_{ij}^k - r_{ji}^k$ and r_{ji}^k is revealed by P_i in the reconstruction phase.
      ii. For every $P_i \in GOOD_k$, let $s_i^k = s_{ji}^k$ for some $P_j \in S_k$.
      iii. Choose any $P_i \in GOOD_k$ and assign $s_k = s_i^k$.
2. Compute D's secret s as $s = \sum_{k=1}^{K} s_k$.

Claim 3.28 *Irrespective of whether D is honest or corrupted, for any subset $S_k \in \overline{\mathcal{B}}$, the following will hold:*

1. *All honest party(ies) in S_k obtain the same share, say s_k, from D in the sharing phase.*
2. *In the reconstruction phase, s_k will be considered as D's commitment to S_k, with probability at least $(1 - \frac{\epsilon}{K})$.*

PROOF: The first part of the claim holds trivially if D is honest. If D is corrupted and has distributed different shares to some pair of honest parties (P_i, P_j) in S_k, then by the working of 4-Round-VSS, (P_i, P_j) will be a **conflicting pair** and D has to broadcast s_k, the share corresponding to the subset S_k. This implies $S_k \in \mathcal{B}$. This is a contradiction to our assumption that $S_k \in \overline{\mathcal{B}}$. Hence we have proved the first part of the claim.

Now we will prove the second part of the claim. Let the honest party(ies) in the set S_k receives s_k from D in sharing phase. We will prove that s_k will be considered as D's commitment to subset S_k in the reconstruction phase with probability at least $(1 - \frac{\epsilon}{K})$, irrespective of whether D is honest or faulty. For that it is enough to show that $GOOD_k$ will not be $\emptyset$ (and will contain at least one party) and for every party $P_i \in GOOD_k$, s_i^k will be equal to s_k with probability at least $(1 - \frac{\epsilon}{K})$.

To assert the statement stated above, we first show that an honest party P_i in S_k will be included in $GOOD_k$ with $s_i^k = s_k$ with probability $(1 - \frac{\epsilon}{nK})$. Note that honest P_i in S_k will be successful in revealing $ICSig(P_j, P_i, \mathcal{P}, r^k_{ji})$ for a $P_j \in S_k$ (with probability one when P_j is honest (by **ICP-Correctness1**); with probability at least $(1 - \frac{\epsilon}{n^2 K})$ when P_j is corrupted (by **ICP-Correctness2**)). In the worst case S_k may contain all the t corrupted parties. Therefore, honest P_i in S_k will be successful in revealing $ICSig(P_j, P_i, \mathcal{P}, r^k_{ji})$ for all $P_j \in S_k$ with probability at least $(1 - t\frac{\epsilon}{n^2 K}) \approx (1 - \frac{\epsilon}{nK})$. Thus we have the following: (a) honest P_i in S_k will be successful in revealing $ICSig(P_j, P_i, \mathcal{P}, r^k_{ji})$ for every $P_j \in S_k$ with probability at least $(1 - \frac{\epsilon}{nK})$; and (b) s^k_{ji} values for all $P_j \in S_k$ will be equal where $s^k_{ji} = b^k_{ij} - r^k_{ji}$ and r^k_{ji} is revealed by P_i. The above implies that an honest party P_i in S_k will be included in $GOOD_k$ with $s^k_i = s_k$ with probability $(1 - \frac{\epsilon}{nK})$.

This proves that $GOOD_k \neq \emptyset$ as there is at least one honest party in a set S_k (that contains $t+1$ parties).

We now show that even a corrupted party $P_i \in GOOD_k$ can ensure $s^k_i = \overline{s_k} \neq s_k$, with probability at most $\frac{\epsilon}{nK}$. Let P_j be an honest party in S_k (possibly the only honest party in S_k). In the sharing phase, P_i had received $ICSig(P_j, P_i, \mathcal{P}, r^k_{ji})$ from P_j. Moreover, b^k_{ij} broadcasted by P_i was equal to $a^k_{ji} = r^k_{ji} + s_k$ broadcasted by P_j (otherwise (P_i, P_j) was a **conflicting pair** which further implies $S_k \in \mathcal{B}$; this is a contradiction). Now in reconstruction phase, P_i can reveal $ICSig(P_j, P_i, \mathcal{P}, \overline{r^k_{ji}})$ with probability $\frac{\epsilon}{n^2 K}$ (by **ICP-Correctness3**) and thus he can ensure $\overline{s^k_{ji}} = b^k_{ij} - \overline{r^k_{ji}}$ to be revealed where $\overline{s^k_{ji}} \neq s_k$. As there can be $\mathcal{O}(t)$ honest parties in S_k, corrupted P_i can ensure that $\overline{s^k_{ji}} = b^k_{ij} - \overline{r^k_{ji}}$ for every honest $P_j \in S_k$, with probability $\mathcal{O}(t)\frac{\epsilon}{n^2 K} \approx \frac{\epsilon}{nK}$. Hence a corrupted party $P_i \in GOOD_k$ can not ensure $s^k_i = \overline{s_k} \neq s_k$, with probability at least $(1 - \frac{\epsilon}{nK})$. Now as there can be $\mathcal{O}(t)$ corrupted parties in S_k, none of them will be able to ensure $s^k_i = \overline{s_k} \neq s_k$ (for different i), with probability at least $(1 - \mathcal{O}(t)\frac{\epsilon}{nK}) \approx (1 - \frac{\epsilon}{K})$. This shows that for every party $P_i \in GOOD_k$, s^k_i will be equal to s_k with probability at least $(1 - \frac{\epsilon}{K})$. This proves that s_k will be considered as D's commitment to subset S_k in the reconstruction phase with probability at least $(1 - \frac{\epsilon}{K})$. □

Lemma 3.29 (Correctness) *Protocol 4-Round-VSS satisfies correctness property, except with error probability ϵ.*

Proof: We have to consider the case when D is honest. For every subset $S_k \in \mathcal{B}$, honest D will correctly broadcast s_k during sharing phase. Also from Claim 3.28, for every $S_k \in \overline{\mathcal{B}}$, honest D's commitment s_k for subset S_k will be recovered correctly with probability at least $(1 - \frac{\epsilon}{K})$. As $|\overline{\mathcal{B}}|$ can be as big as K, D's commitment s_k for every S_k in $|\overline{\mathcal{B}}|$ will be reconstructed correctly, with probability at least $(1 - K\frac{\epsilon}{K}) = (1 - \epsilon)$. So D's secret $s = \sum_{k=1}^{K} s_k$ will be reconstructed correctly with probability at least $(1 - \epsilon)$. □

Lemma 3.30 (Strong Commitment) *Protocol 4-Round-VSS satisfies strong commitment property, except with error probability ϵ.*

Proof: We have to consider the case when D is corrupted. Now in the sharing phase D's commitment to a subset S_k is as follows:

1. If $S_k \in \mathcal{B}$: The s_k broadcasted by D during the sharing phase.
2. If $S_k \in \overline{\mathcal{B}}$: The common value s_k received from D by all the honest party(ies) during sharing phase (from Claim 3.28, all honest parties in $S_k \in \overline{\mathcal{B}}$ receive same s_k from D).

So the unique secret s committed by D during sharing phase is the sum of s_k values over all the subsets. Now by Claim 3.28, for an $S_k \in \overline{\mathcal{B}}$, D's commitment to S_k i.e s_k will be recovered correctly with probability at least $(1 - \frac{\epsilon}{K})$. Now it is easy to see that D's committed secret s will be reconstructed in the reconstruction phase with probability at least $(1 - \epsilon)$ (following the argument given in Lemma 3.29). □

Theorem 3.31 *There exists an 4-round sharing, 2-round reconstruction $(2t + 1, t)$ statistical VSS protocol.*

PROOF: Protocol 4-Round-VSS achieves correctness and strong commitment, except with error probability ϵ and also achieves perfect secrecy. This follows from Lemma 3.27, 3.29 and 3.30. □

It is to be noted that the number of rounds in reconstruction phase of 4-Round-VSS is optimal from the results of [49]. But we can have a 4-round sharing 1-round reconstruction VSS if we consider the adversary to be non-rushing (as shown in the next section).

3.6.1 4-round Sharing VSS with One Round of Reconstruction

As in 3-Round-VSS, if we restrict the adversary to a non-rushing adversary then the two rounds of reconstruction of protocol 4-Round-VSS can be collapsed into a single round. Hence, we have the following theorem:

Theorem 3.32 *If the adversary is non-rushing then there exists an in-efficient 4-round sharing 1-round reconstruction $(2t + 1, t)$ statistical VSS protocol.*

3.7 Efficient 5-round Sharing, 2-round Reconstruction $(2t+1, t)$ Statistical VSS

We defer the presentation of our efficient 5-round sharing and 2-round reconstruction $(2t + 1, t)$ statistical VSS scheme in the next Chapter (in Section 4.2 and 4.3). Our protocol will use IC signatures.

As in 3-Round-VSS and 4-Round-VSS, the number of rounds in reconstruction phase of our 5-round sharing VSS is optimal from the results of [49]. But again we can have a 5-round sharing 1-round reconstruction VSS if we consider the adversary to be non-rushing.

3.7.1 5-round Sharing VSS with One Round of Reconstruction

As in 3-Round-VSS and 4-Round-VSS, if we restrict the adversary to a non-rushing adversary then the two rounds of reconstruction phase of 5-round sharing VSS can be collapsed into a single round. This is because, the reconstruction phase of our VSS will consist of revelations of IC signatures which can be collapsed into a single round in the presence of non-rushing adversary. Hence, we have the following theorem:

Theorem 3.33 *If the adversary is non-rushing then there exists an efficient 5-round sharing 1-round reconstruction $(2t + 1, t)$ statistical VSS protocol.*

3.8 Lower Bounds for Statistical VSS

3.8.1 Lower Bound for 2-round Sharing Statistical VSS

We now prove the optimality of our 2-round sharing $(3t+1, t)$ statistical VSS protocol, with respect to the resilience.

Theorem 3.34 *There is no 2-round sharing (n, t) statistical VSS protocol with $n \leq 3t$, irrespective of the number of rounds in the reconstruction phase.*

In fact we prove the following stronger result from which the above theorem follows immediately.

Theorem 3.35 *There is no 2-round sharing (n, t) statistical WSS protocol with $n \leq 3t$, irrespective of the number of rounds in the reconstruction phase.*

To prove the above theorem, we use standard player partitioning arguments [91] and prove the following lemma:

Lemma 3.36 *There is no 2-round sharing $(3, 1)$ statistical WSS protocol, irrespective of the number of rounds in the reconstruction phase.*

We now prove Lemma 3.36 by contradiction. Let the set of parties be $\{P_1, P_2, P_3\}$ with $D = P_1$, and assume there exists a 2-round sharing protocol Π for statistical WSS. Without loss of generality we assume that messages from round 2 onwards in Π are broadcasted. This holds without loss of generality since the parties can exchange random pads in the first round and then they can use these random pads to unmask broadcasts in later rounds (this is a well known result [91]).

Let us now look at the structure of the sharing phase of Π. Let party P_i start with random coin r_i[3]. In the first round, private messages are exchanged between parties and also parties broadcast messages individually. The private messages and broadcast messages of P_i are function of its random coin r_i. We denote the private message that P_i sent to P_j by r_{ij}, and the broadcast of party P_i by α_i. So given r_i, we assume that P_i's round 1 private messages can be deterministically generated. Similarly, we may write $\alpha_i(r_i)$ to mean that given r_i, the broadcast message α_i can be generated deterministically. Now recall that round 2 messages are all broadcasts. Let the broadcast by party P_i in the *second* round be denoted by β_i. Technically, β_is are functions of $r_i, \{r_{ji}\}, \{\alpha_j(r_j)\}$ (for $j \neq i$). So we may write $\beta_i(r_i, \{r_{ji}\}, \{\alpha_j(r_j)\}$ (for $j \neq i$)). At the end of the second round, each party locally outputs his *view* of the sharing phase i.e all the information (broadcasted as well as private) seen by that party so far. Following is the formal definition of view V_i of a party P_i in protocol Π.

Definition 3.37 *The* view *of a party P_i denoted by V_i in protocol* Π *consists of the random coin r_i of P_i and all the messages (private messages and broadcasts) received by him during the sharing phase of* Π.

The formal description of the *sharing phase* of protocol Π is given below:

---

1. P_1, P_2 and P_3 participate in protocol Π with random coins r_1, r_2 and r_3, respectively. D has input s (implicitly defined by r_1).

[3] r_i's are actually random variables here. For different executions of Π, they may take different values.

2. Round 1

   (a) Private messages: $r_{12}, r_{13}, r_{21}, r_{23}, r_{31}, r_{32}$.

   (b) Broadcasts: $\alpha_1(r_1), \alpha_2(r_2), \alpha_3(r_3)$.

3. Round 2 broadcasts: $\beta_1(r_1, r_{21}, r_{31}, \alpha_2(r_2), \alpha_3(r_3))$,
   $\beta_2(r_2, r_{12}, r_{32}, \alpha_1(r_1), \alpha_3(r_3))$,
   $\beta_3(r_3, r_{13}, r_{23}, \alpha_1(r_1), \alpha_2(r_2))$.

4. Local outputs:

   (a) $V_1 = (r_1, r_{21}, r_{31}, \alpha_2, \alpha_3, \beta_2, \beta_3)$.

   (b) $V_2 = (r_2, r_{12}, r_{32}, \alpha_1, \alpha_3, \beta_1, \beta_3)$.

   (c) $V_3 = (r_3, r_{13}, r_{23}, \alpha_1, \alpha_2, \beta_1, \beta_2)$.

Without loss of generality, we assume that dealer's input s is implicitly contained in r_1 (i.e., the dealer's random coins). So far we have discussed about the structure of the sharing phase of protocol Π.

Now let us fix how the reconstruction phase of Π would look like. According to the definition of WSS protocol, the reconstruction phase can be simulated by a function, say REC, which takes the views of the parties generated at the end of sharing phase. In other words, given the views of the parties at the end of the sharing phase, we can always define a function REC to simulate the actual reconstruction phase (that may require any number of rounds in our context). Let us now define REC formally.

Definition 3.38 *The reconstruction function REC takes as input the set of views of all the parties that participate in the reconstruction phase of protocol Π and outputs (a) D's committed secret when D is honest; (b) D's committed secret or $NULL$, when D is corrupted. Since all the honest parties participate in the reconstruction phase, REC will have at least 2 input views. The corrupted parties may input anything as their view. Let $V_H = \{V_i | P_i \text{ is honest}\}$ and let $V_C = \{V_i | P_i \text{ is corrupted}\}$. Let s be the fixed secret that D is committed to in the sharing phase. Then REC satisfies the following with very high probability,*

- *For every possible value of V_C, $\mathsf{REC}(V_H, V_C) = s$ when D is honest (from correctness property) and $\mathsf{REC}(V_H, V_C) = s/NULL$ when D is corrupted (from weak-commitment property).*

For our purpose, we allow REC to internally simulate the behavior of all the parties in the actual reconstruction phase of Π. That is, REC assumes that all the parties (including those that deviated from the protocol in the sharing phase) act honestly in the reconstruction phase. Of course this assumption does not stop a corrupted party to input junk view to REC. What we mean by the previous statements is that once all the inputs are fed to REC function, REC internally simulates the honest behavior of the parties with the inputs.

We will now describe a real execution G of Π, where D is corrupted though he does not behave that way during the communication of sharing phase. The only corrupted behavior that D shows is when he inputs his view to REC. We show that G does not satisfy weak commitment property. That is depending on the view that D inputs to REC, the reconstructed secret may change. This will in turn show that Π does not satisfy weak commitment property. This is because we prove the above by considering an arbitrary execution G of Π. *That is G can be any execution of Π, where the break in weak commitment is possible.*

The sharing phase of execution G is as follows, where D *honestly* follows the steps of Π (though corrupted). We may denote the view of P_i by $V_i(G)$ in execution G.

---

1. P_1, P_2 and P_3 participate in G with random coins r_1^G, r_2^G and r_3^G, respectively. D has input s^G (implicitly defined by r_1^G).
2. Round 1
   (a) Private messages: $r_{12}^G, r_{13}^G, r_{21}^G, r_{23}^G, r_{31}^G, r_{32}^G$.
   (b) Broadcasts: $\alpha_1(r_1^G), \alpha_2(r_2^G), \alpha_3(r_3^G)$.
3. Round 2 broadcasts: $\beta_1(r_1^G, r_{21}^G, r_{31}^G, \alpha_2(r_2^G), \alpha_3(r_3^G))$,
   $\beta_2(r_2^G, r_{12}^G, r_{32}^G, \alpha_1(r_1^G), \alpha_3(r_3^G))$,
   $\beta_3(r_3^G, r_{13}^G, r_{23}^G, \alpha_1(r_1^G), \alpha_2(r_2^G))$.
4. Local outputs:
   (a) $V_1(G) = (r_1^G, r_{21}^G, r_{31}^G, \alpha_2, \alpha_3, \beta_2, \beta_3)$.
   (b) $V_2(G) = (r_2^G, r_{12}^G, r_{32}^G, \alpha_1, \alpha_3, \beta_1, \beta_3)$.
   (c) $V_3(G) = (r_3^G, r_{13}^G, r_{23}^G, \alpha_1, \alpha_2, \beta_1, \beta_2)$.

Now we claim the following:

Claim 3.39 *$REC(V_1^\star(G), V_2^\star(G), \clubsuit) = s^G$, with very high probability, where*

1. $V_1^\star(G) = (r_1^G, r_{21}^G, r_{31}^G, \alpha_2, \alpha_3, \beta_2, \beta_3^\star)$
2. $V_2^\star(G) = (r_2^G, r_{12}^G, r_{32}^G, \alpha_1, \alpha_3, \beta_1, \beta_3^\star)$

and $\clubsuit$ can be anything including $V_3(G)$ and $\beta_3^\star$ can be anything including β_3.

PROOF: To show that our claim is true, let us consider an execution G' where D is honest and P_3 is corrupted; P_1, P_2 and P_3 have the same random coins as in execution G. But in round 2, corrupted P_3 broadcasts $\beta_3^\star$ which can be anything including β_3. So the sharing phase of G' looks as follows:

---

1. P_1, P_2 and P_3 participate in G with random coins r_1^G, r_2^G and r_3^G, respectively. D has input s^G (implicitly defined by r_1^G).
2. Round 1
   (a) Private messages: $r_{12}^G, r_{13}^G, r_{21}^G, r_{23}^G, r_{31}^G, r_{32}^G$.
   (b) Broadcasts: $\alpha_1(r_1^G), \alpha_2(r_2^G), \alpha_3(r_3^G)$.

3. Round 2 broadcasts: $\beta_1(r_1^G, r_{21}^G, r_{31}^G, \alpha_2(r_2^G), \alpha_3(r_3^G))$,
   $\beta_2(r_2^G, r_{12}^G, r_{32}^G, \alpha_1(r_1^G), \alpha_3(r_3^G))$,
   $\beta_3^\star$.

4. Local outputs:

   (a) $V_1(G') = (r_1^G, r_{21}^G, r_{31}^G, \alpha_2, \alpha_3, \beta_2, \beta_3^\star)$.
   (b) $V_2(G') = (r_2^G, r_{12}^G, r_{32}^G, \alpha_1, \alpha_3, \beta_1, \beta_3^\star)$.
   (c) $V_3(G') = \clubsuit$ ($\clubsuit$ can be anything including $V_3(G)$).

Now by correctness property of statistical WSS, $\mathsf{REC}(V_1(G'), V_2(G'), V_3(G')) = s^G$ with very high probability. This shows that our claim is true. $\square$

Now let us consider another execution H which we show to be possible always as otherwise, we can prove that Π breaches perfect secrecy. In H, $D = P_1$ starts with some random coin r_1^H that implicitly defines secret $s^H \neq s^G$ and satisfies $r_{12}^H = r_{12}^G$ and $\alpha_1(r_1^H) = \alpha_1(r_1^G)$. P_2 has the same random coin r_2^G as in execution G. P_3 has r_3^H such that $r_{32}^H = r_{32}^G$, $\alpha_3(r_3^H) = \alpha_3(r_3^G)$ and $\beta_1(r_1^H, r_{21}^G, r_{31}^H, \alpha_2(r_2^G), \alpha_3(r_3^H)) = \beta_1(r_1^G, r_{21}^G, r_{31}^G, \alpha_2(r_2^G), \alpha_3(r_3^G))$. The sharing phase of H is described as follows:

---

1. P_1, P_2 and P_3 participate in G with random coins r_1^H, r_2^G and r_3^H, respectively. D has input s^H (implicitly defined by r_1^H).

2. Round 1

   (a) Private messages: $r_{12}^H = r_{12}^G, r_{13}^H, r_{21}^G, r_{23}^G, r_{31}^H, r_{32}^H = r_{32}^G$.
   (b) Broadcasts: $\alpha_1(r_1^H) = \alpha_1(r_1^G), \alpha_2(r_2^G), \alpha_3(r_3^H) = \alpha_3(r_3^G)$.

3. Round 2 broadcasts: $\beta_1(r_1^H, r_{21}^G, r_{31}^H, \alpha_2, \alpha_3) = \beta_1(r_1^G, r_{21}^G, r_{31}^G, \alpha_2, \alpha_3)$,
   $\beta_2(r_2^G, r_{12}^G, r_{32}^G, \alpha_1, \alpha_3)$,
   $\overline{\beta_3}(r_3^H, r_{13}^H, r_{23}^G, \alpha_1, \alpha_2)$. *$\overline{\beta_3}$ may or may not be equal to β_3.*

4. Local outputs:

   (a) $V_1(H) = (r_1^H, r_{21}^G, r_{31}^H, \alpha_2, \alpha_3, \beta_2, \overline{\beta_3})$.
   (b) $V_2(H) = (r_2^G, r_{12}^G, r_{32}^G, \alpha_1, \alpha_3, \beta_1, \overline{\beta_3})$.
   (c) $V_3(H) = (r_3^H, r_{13}^H, r_{23}^G, \alpha_1, \alpha_2, \beta_1, \beta_2)$.

We claim that H is a possible execution of Π. For this we just have to prove that for every r_1^H (satisfying the properties mentioned above), there is always a random coin r_3^H that ensures $\beta_1(r_1^H, r_{21}^G, r_{31}^H, \alpha_2, \alpha_3) = \beta_1(r_1^G, r_{21}^G, r_{31}^G, \alpha_2, \alpha_3)$ along with other constraints as mentioned above. We prove this in the following claim:

Claim 3.40 *For every r_1^H representing a secret $s^H \neq s^G$, there is always a random coin r_3^H that ensures $\beta_1(r_1^H, r_{21}^G, r_{31}^H, \alpha_2, \alpha_3) = \beta_1(r_1^G, r_{21}^G, r_{31}^G, \alpha_2, \alpha_3)$.*

PROOF: We prove the claim by contradiction. Let for r_1^H there is no such r_3^H that can ensure $\beta_1(r_1^H, r_{21}^G, r_{31}^H, \alpha_2, \alpha_3) = \beta_1(r_1^G, r_{21}^G, r_{31}^G, \alpha_2, \alpha_3)$. This implies that whatever may be the value of r_{31}^H, the above equation will never hold for r_{31}^H. This clearly violates the secrecy. This can be argued as follows. Let P_2 be the corrupted party in execution G described before and assume that it behaves passively throughout. Then at the end of reconstruction, he will know that the secret is s^G for the execution G. Now he participates in H and sees that $\overline{\beta_1}$ (different from β_1) has been broadcasted by D as opposed to β_1 in G. This allows P_2 to guess that in H, s^G is not the secret that is shared. This breaches the perfect secrecy. □

So we have proved that execution H is a possible execution of Π. Now we show the following:

Claim 3.41 *$\mathsf{REC}(V_1^\star(H), V_2^\star(G), \clubsuit) = s^H$, with very high probability, where*

1. *$V_1^\star(H) = (r_1^H, r_{21}^G, r_{31}^H, \alpha_2, \alpha_3, \beta_2, \beta_3^\star)$.*
2. *$V_2^\star(G) = (r_2^G, r_{12}^G, r_{32}^G, \alpha_1, \alpha_3, \beta_1, \beta_3^\star)$.*

and $\clubsuit$ can be anything including $V_3(H)$; and $\beta_3^\star$ can be anything including $\overline{\beta_3}$.

PROOF: This can be shown following the same approach as used in Claim 3.39. □

We again stress that the above two claims (i.e Claim 3.40 and 3.41) hold for any choice of r_1^H and r_3^H satisfying the constraints mentioned before. Now we are in a situation to show how corrupted D may give different inputs to REC in execution G to force reconstruction of different secrets with very high probability.

Lemma 3.42 *In execution G, the dealer D may give different inputs to REC to force reconstruction of different secrets with very high probability.*

PROOF: D plays the following mental game after the completion of the sharing phase of G. D selects some r_1^H such that it implicitly defines secret $s^H \neq s^G$ and satisfies $r_{12}^H = r_{12}^G$ and $\alpha_1(r_1^H) = \alpha_1(r_1^G)$. D also correspondingly finds r_{31}^H such that $\beta_1(r_1^H, r_{21}^G, r_{31}^H, \alpha_2(r_2^G), \alpha_3(r_3^H)) = \beta_1(r_1^G, r_{21}^G, r_{31}^G, \alpha_2(r_2^G), \alpha_3(r_3^G))$. By our argument in Claim 3.40, there will be some r_3^H such that the r_{31}^H will make the above equality hold. Note that D can always find such r_{31}^H by solving the equation (for x) $\beta_1(r_1^H, r_{21}^G, x, \alpha_2(r_2^G), \alpha_3(r_3^H)) = \beta_1(r_1^G, r_{21}^G, r_{31}^G, \alpha_2(r_2^G), \alpha_3(r_3^G))$. Now in the reconstruction phase, if D inputs his view as $V_1(G) = (r_1^G, r_{21}^G, r_{31}^G, \alpha_2, \alpha_3, \beta_2, \beta_3)$, then at a glance the input views to REC are as follows:

1. $V_1(G) = (r_1^G, r_{21}^G, r_{31}^G, \alpha_2, \alpha_3, \beta_2, \beta_3)$.
2. $V_2(G) = (r_2^G, r_{12}^G, r_{32}^G, \alpha_1, \alpha_3, \beta_1, \beta_3)$.
3. $V_3(G) = (r_3^G, r_{13}^G, r_{23}^G, \alpha_1, \alpha_2, \beta_1, \beta_2)$.

Now in Claim 3.39, we may replace $\beta_3^\star$ by β_3 and $\clubsuit$ by $V_3(G)$ and therefore claim that s^G will be reconstructed with very high probability.

On the other hand, if D inputs his view as $V_1(G) = (r_1^H, r_{21}^G, r_{31}^H, \alpha_2, \alpha_3, \beta_2, \beta_3)$, then at a glance the input views to REC are as follows:

1. $V_1(G) = (r_1^H, r_{21}^G, r_{31}^H, \alpha_2, \alpha_3, \beta_2, \beta_3)$.
2. $V_2(G) = (r_2^G, r_{12}^G, r_{32}^G, \alpha_1, \alpha_3, \beta_1, \beta_3)$.

3. $V_3(G) = (r_3^G, r_{13}^G, r_{23}^G, \alpha_1, \alpha_2, \beta_1, \beta_2)$.

Now in Claim 3.41, we may replace $\beta_3^\star$ by β_3 and $\clubsuit$ by $V_3(G)$ and therefore claim that s^H will be reconstructed with very high probability. Thus we have shown that a corrupted D can always force during the reconstruction phase the output of the protocol to be one of two secrets, thus violating the weak commitment property. $\square$

From the above proof, we conclude that there does not exist a 2-round sharing $(3, 1)$ statistical WSS and hence 2-round sharing $(3t, t)$ statistical WSS and finally 2-round sharing $(3t, t)$ statistical VSS protocol, with any number of rounds in the reconstruction phase. This implies that there does not exist a 2-round sharing (n, t) statistical VSS (and WSS) protocol with $n \leq 3t$, for any number of rounds in the reconstruction phase.

3.8.2 Lower Bound for 1-round Sharing Statistical VSS

We now derive a non-trivial lower bound on the fault tolerance of any 1-round sharing statistical VSS (with any number of rounds in reconstruction phase).

Theorem 3.43 *1-round sharing statistical VSS is possible only if* $((t = 1)$ *and* $(n \geq 4))$*, irrespective of the number of rounds in reconstruction phase.*

PROOF: The impossibility of 1-round sharing $(3, 1)$ statistical VSS with any number of rounds in reconstruction, follows from Theorem 3.36, where it is proved that VSS with 2-round sharing (and any number of rounds in reconstruction phase) is impossible for $n \leq 3t$ (putting $t = 1$, we get our impossibility). Now we show that for $t \geq 2$ there does not exist any 1-round sharing (n, t) statistical VSS protocol with $n \geq 4$, irrespective of the number of rounds in the reconstruction phase. We prove the above statement assuming $t = 2$ in Lemma 3.44.

Lemma 3.44 *There does not exist any 1-round sharing* $(n, 2)$ *statistical VSS protocol with* $n \geq 4$*, irrespective of the number of rounds in the reconstruction phase.*

PROOF: We now prove this lemma by contradiction. Let the set of parties be $\{P_1, \ldots, P_n\}$, and assume there exists a 1-round sharing protocol Π for statistical VSS with D being any party other than P_1 (this can be assumed without loss of generality).

Let us now look at the structure of the sharing phase of Π. For this we will almost stick to the notations used in Subsection 3.8.1 for 2-round sharing WSS protocol Π. Let party P_i start with random coin r_i[4]. In the first round, private messages are exchanged between parties and also parties broadcast messages individually. The private messages and broadcast messages of P_i are function of its random coin r_i. We denote the private message that P_i sends to P_j by r_{ij}, and the broadcast of party P_i by α_i. So given r_i, we assume that P_i's round 1 private messages can be deterministically generated. Similarly, we may write $\alpha_i(r_i)$ to mean that given r_i, the broadcast message α_i can be generated deterministically. At the end of the sharing phase, each party locally outputs his *view* of the sharing

[4] r_i's are actually random variables here. For different executions of Π, they may take different values.

phase i.e all the information (broadcasted as well as private) seen by that party so far (definition of view can be found in Definition 3.37 of subsection 3.8.1).

The formal description of the *sharing phase* of Π is given below:

---

1. $P_1, \ldots, P_n$ participate in protocol Π with random coins $r_1, \ldots, r_n$, respectively. D has input s (implicitly defined by r_D[5], where $r_D = r_i$ if P_i is D).
2. Round 1
   (a) Private messages communicated by the parties.
      i. Private messages of P_1: $r_{12}, r_{13}, \ldots, r_{1n}$.
      ii. ....
      iii. Private messages of P_n: $r_{n1}, r_{n2}, \ldots, r_{n(n-1)}$.
   (b) Broadcasts: $\alpha_1(r_1), \ldots, \alpha_n(r_n)$.
3. Local outputs:
   (a) $V_1 = (r_1, r_{21}, \ldots, r_{n1}, \alpha_2, \ldots, \alpha_n)$.
   (b) $V_2 = (r_2, r_{12}, r_{32}, \ldots, r_{n2}, \alpha_1, \alpha_3, \ldots, \alpha_n)$.
   (c) ......
   (d) $V_n = (r_n, r_{1n}, \ldots, r_{(n-1)n}, \alpha_1, \ldots, \alpha_{n-1})$.

Without loss of generality, we assume that dealer's secret s is implicitly contained in r_D (i.e., the dealer's random coin). So far we have discussed about the structure of the sharing phase of protocol Π.

Now let us fix how a reconstruction phase of Π would look like. According to the definition of VSS protocol, the reconstruction phase can be simulated by a function, say REC, which takes the views of the parties generated at the end of sharing phase. In other words, given the views of the parties at the end of the sharing phase, we can always define a function REC to simulate the actual reconstruction phase (that may require any number of rounds in our context). Let us now define REC formally[6].

Definition 3.45 *The reconstruction function REC takes as input the set of views of all the parties that participate in the reconstruction phase of protocol Π and outputs D's committed secret irrespective of whether D is honest or corrupted. Since all the honest parties participate in the reconstruction phase, REC will have at least $n-2$ input views. The corrupted parties may input anything as their view. Let $V_H = \{V_i | P_i$ is honest$\}$ and let $V_C = \{V_i | P_i$ is corrupted$\}$. Let s be the fixed secret that D is committed to in the sharing phase. Then REC satisfies the following with very high probability,*

[5] From now onwards we distinguish D's random coin by r_D.

[6] This definition of REC will be slightly different from the one presented in subsection 3.8.1. This is because in the current section we are dealing with VSS, whereas WSS was dealt with in subsection 3.8.1.

- *For every possible value of* V_C, $\mathsf{REC}(V_H, V_C) = s$ *(follows from correctness property when* D *is honest; follows from strong commitment property when* D *is corrupted).*

For our purpose, we allow REC to internally simulate the behavior of all the parties in the actual reconstruction phase of Π. That is, REC assumes that all the parties (including those that deviated from the protocol in the sharing phase) act honestly in the reconstruction phase. Of course this assumption does not stop a corrupted party to input junk view to REC. What we mean by the previous statements is that once all the inputs are fed to REC function, REC internally simulates the honest behavior of the parties with the inputs.

We now start with a real execution G of Π where D's secret is s^G. We may denote the view of P_i by $V_i(G)$ in execution G.

---

1. $P_1, \ldots, P_n$ participate in execution G with random coins $r_1^G, \ldots, r_n^G$, respectively. D has input s^G (implicitly defined by r_D^G).
2. Round 1
   (a) Private messages communicated by the parties.
      i. Private messages of P_1: $r_{12}^G, r_{13}^G, \ldots, r_{1n}^G$.
      ii. ....
      iii. Private messages of P_n: $r_{n1}^G, r_{n2}^G, \ldots, r_{n(n-1)}^G$.
   (b) Broadcasts: $\alpha_1(r_1^G), \ldots, \alpha_n(r_n^G)$.
3. Local outputs:
   (a) $V_1(G) = (r_1^G, r_{21}^G, \ldots, r_{n1}^G, \alpha_2^G, \ldots, \alpha_n^G)$.
   (b) $V_2(G) = (r_2^G, r_{12}^G, r_{32}^G, \ldots, r_{n2}^G, \alpha_1^G, \alpha_3^G, \ldots, \alpha_n^G)$.
   (c) ......
   (d) $V_n = (r_n^G, r_{1n}^G, \ldots, r_{(n-1)n}^G, \alpha_1^G, \ldots, \alpha_{n-1}^G)$.

By the property of REC, we have the following claim:

Claim 3.46 $\mathsf{REC}(V_1(G), \ldots, V_n(G)) = s^G$, *with very high probability.*

Let $V_i^\star(G)$ is defined to be same as $V_i(G)$ with r_{Di}^G is replaced by any value $\overline{r_{Di}^G}$. Now we show the following:

Claim 3.47 $\mathsf{REC}(V_1(G), \ldots, V_n^\star(G)) = s^G$, *with very high probability.*

PROOF: Let in G, the dealer D was honest and P_n was corrupted. At the end of sharing phase, let P_n replaces r_{Dn}^G (that he has received from D) by any value $\overline{r_{Dn}^G}$ in his view $V_n(G)$ and inputs it to REC. By correctness of Π, function REC should output s^G with very high probability. This proves our claim. □

Claim 3.48 $\mathsf{REC}(V_1(G), \ldots, V_{n-1}^\star(G), V_n^\star(G)) = s^G$ *with very high probability.*

PROOF: Let in G, the dealer D was corrupted and distributed r_{Di}^G for all $i = 1, \ldots, n-1$ and $\overline{r_{Dn}^G}$ (this can be any value) to P_n. Now if every party (including D) behaves properly and inputs correct views then by Claim 3.47, s^G will be reconstructed. Now on the other hand, let P_{n-1} becomes corrupted at the end of sharing phase (apart from D) and replaces $r_{D(n-1)}^G$ (that he has received from D) by any value $\overline{r_{D(n-1)}^G}$ in his view $V_{n-1}(G)$ and inputs it to REC. By strong commitment property of Π, function REC should still output s^G with very high probability. This shows that our claim is true. □

Like this we can proceed and prove the following claim.

Claim 3.49 *$\mathsf{REC}(V_1(G), V_2^\star(G), \ldots, V_{n-1}^\star(G), V_n^\star(G)) = s^G$ with very high probability.*

Finally the above Claim clearly shows a violation of the secrecy property of Π because it states that in any execution, where D gives message r_{D1}^G to P_1, will always output the secret s^G at the end of the reconstruction phase. So if D is honest and adversary passively corrupts P_1 in such an execution, he will come to know that the shared secret is s^G, which is a violation of perfect secrecy property. Lemma 3.44 now follows from the above discussion. □

Note that the above proof for Lemma 3.44 does not hold for WSS due to the fact that WSS requires only weak commitment, this prevents the argument that all sequences of messages sent to the parties need to be reconstructed to the same secret. In fact we can design a 1-round sharing, 2-round reconstruction $(3t+1, t)$ statistical WSS protocol. The protocol is presented in the next section.

3.9 Efficient 1-round Sharing, 2-round Reconstruction $(3t+1, t)$ Statistical WSS

We now design a 1-round sharing, 2-round reconstruction $(3t+1, t)$ statistical WSS protocol. This shows that the bound given in Theorem 3.43 does not hold for 1-round sharing statistical WSS. In perfect settings, any two or less round sharing WSS protocol requires at least $4t+1$ parties [73]. Therefore, we see that even in the case of WSS, probabilistically relaxing the conditions helps to improve fault tolerance.

The Intuition: In the sharing phase of our WSS protocol (named as 1-Round-WSS), D picks $n+1$ random polynomials $F(x), f_1(x), \ldots, f_n(x)$ of degree t such that $F(0)$ is the secret s and $f_i(0) = F(i)$ for $i = 1, \ldots, n$. D also selects n random non-zero secret evaluation points $\alpha_1, \ldots, \alpha_n$. To P_i, D delivers α_i, $f_i(x)$ and values of all the $f_j(x)$ polynomials at α_i.

In the reconstruction phase, the parties reveal their polynomials and the values (along with the secret evaluation point that they have) in **Round 1** and **Round 2** respectively. We then define two types of parties called *affirmed* and *semi-affirmed*. A party P_i will be called as *affirmed* if there are at least $2t+1$ parties whose values have matched with P_i's broadcasted polynomial. Like wise, a party P_i will be called as *semi-affirmed* if there are at least $t+1$ parties and at most $2t$ parties whose values have matched with P_i's broadcasted polynomial. It is easy to show that *when D is honest*, then all the honest parties will be considered as *affirmed* (as their polynomials will match with the values broadcasted by all the $2t+1$ honest parties) and no party can be considered as *semi-affirmed* with high probability. The latter can be argued as follows: A corrupted party can rarely broadcast a changed polynomial in **Round 1** (other than what was handed over to him by D in the sharing phase) that can match with the values of an honest party broadcasted only in **Round 2**. Now in the reconstruction phase, a NULL will be reconstructed when there are less than $2t+1$ *affirmed* parties or there is at least one *semi-affirmed* party. Finally if the constant terms of the broadcasted polynomials of the *affirmed* parties define a degree t polynomial $F(x)$, then $F(0)$ is considered as the recovered secret. Otherwise, again NULL will be reconstructed. The protocol is given in Fig. 3.7.

Our protocol has an error probability of ϵ. To bound the error probability by ϵ, the computation in our statistical WSS protocol is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^2 2^{-\kappa}$. So we have $|\mathbb{F}| \geq \frac{n^2}{\epsilon}$.

Lemma 3.50 (Secrecy) *Protocol 1-Round-WSS satisfies perfect secrecy.*

PROOF: We have to consider the case when D is honest. Without loss of generality, let $\mathcal{A}_t$ controls the first t parties during sharing phase. Then $\mathcal{A}_t$ knows $f_1(x), \ldots, f_t(x)$ and hence $f_1(0), \ldots, f_t(0)$, which is insufficient to know $F(x)$ and hence $F(0)$. Adversary will also know t distinct points on each $f_i(x)$. The points on $f_1(x), \ldots, f_t(x)$ are already known to $\mathcal{A}_t$ and can be removed from his view. Since degree of each $f_i(x)$ is t, adversary lacks one point on each $f_{t+1}(x), \ldots, f_n(x)$ to completely know them and hence information theoretic security on $F(0) = s$ holds. □

Claim 3.51 *If D is honest, then a corrupted P_i producing $f_i'(x) \neq f_i(x)$ in* **Reconstruction Phase** *will be accepted by an honest P_j with probability at most $\frac{\epsilon}{n^2}$.*

PROOF: The proof follows from the fact that P_i produces $f_i'(x) \neq f_i(x)$ in **Round 1** of **Reconstruction Phase** without knowing α_j, v_{ij}, corresponding to honest P_j. So P_i will be *accepted* by an honest P_j only if P_i can correctly guess α_j such that $f_i'(\alpha_j) = f_i(\alpha_j) = v_{ij}$, which can happen with probability at most $\frac{1}{|\mathbb{F}|} \leq \frac{\epsilon}{n^2}$ in our context. □

Lemma 3.52 (Correctness) *Protocol 1-Round-WSS satisfies correctness property, except with error probability ϵ.*

PROOF: We have to consider the case when D is honest. Notice that if D is honest, then all the honest parties (at least $2t+1$) will *accept* each other and each honest P_i will be *affirmed*. If some corrupted P_i produces incorrect $f_i'(x) \neq f_i(x)$, then from Claim 3.51, it can be *accepted* by an honest P_j with probability at most $\frac{\epsilon}{n^2}$. So some honest party may *accept* P_i, with probability at most $(2t+1)\frac{\epsilon}{n^2} \approx \frac{\epsilon}{n}$. This implies a corrupted P_i who produces incorrect $f_i'(x) \neq f_i(x)$, will be *accepted* by at most t corrupted parties and hence P_i will be neither *semi-affirmed* nor *affirmed* with probability at least $(1 - \frac{\epsilon}{n})$. Now there are t corrupted parties who may broadcast wrong polynomials. So the probability that none of them will be considered as *semi-affirmed* or *affirmed* is $(1 - t\frac{\epsilon}{n}) \approx (1 - \epsilon)$.

Figure 3.7: A 1-Round Sharing 2-Round Reconstruction $(3t+1, t)$ Statistical WSS

1-Round-WSS$(D, \mathcal{P}, s, \epsilon)$

Sharing Phase: One Round

1. D picks a random polynomial $F(x)$ over $\mathbb{F}$ of degree t, such that $F(0) = s$.
2. D chooses and sends to P_i the following:
   (a) A random polynomial $f_i(x)$ over $\mathbb{F}$ of degree t, such that $f_i(0) = F(i)$.
   (b) A random non-zero element from $\mathbb{F}$ denoted by α_i (all α_i's are distinct).
   (c) The values $[v_{1i}\ v_{2i}\ \ldots\ v_{ni}]$ such that $v_{ji} = f_j(\alpha_i)$.

Reconstruction Phase: Two Rounds

Round 1: Each $P_i \in \mathcal{P}$ broadcasts polynomial $f'_i(x)$.

Round 2: Each $P_i \in \mathcal{P}$ broadcasts α'_i and n values $[v'_{1i}\ v'_{2i}\ \ldots\ v'_{ni}]$.

1. Party P_i is *accepted* by party P_j if $v'_{ij} = f'_i(\alpha'_j)$.

2. Party P_i is called *affirmed* if it is *accepted* by at least $2t+1$ parties (possibly including itself) where as P_i is called *semi-affirmed* if it is accepted by at least $t+1$ and by at most $2t$ parties (possibly including itself).

Local Computation (By Every Party):

1. IF the number of *affirmed* parties is less than $2t+1$ OR the number of *semi-affirmed* parties is more than zero, then output $NULL$.
2. ELSE consider $f'_i(0)$'s of all the *affirmed* parties and check whether they interpolate a unique t degree polynomial, say $F'(x)$. IF yes, then output $s' = F'(0)$, ELSE output $NULL$.

Therefore, with probability $(1-\epsilon)$, number of *semi-affirmed* parties will be zero and furthermore with the same probability all the corrupted parties who are *affirmed* have broadcasted the correct polynomial (the one received from D in sharing phase). Now it is easy to see that $F(x)$ will be reconstructed correctly and $F(0) = s$ will be the output, with probability $(1-\epsilon)$. □

Claim 3.53 *If D is corrupted and the number of affirmed parties is at least $2t+1$, then at the end of the* **Sharing Phase** *of 1-Round-WSS, there was a unique secret $s^* \in \mathbb{F} \cup \{NULL\}$ defined by $f'_i(0)$ values of the honest affirmed parties.*

PROOF: Since the number of *affirmed* parties is at least $2t+1$, there are at least $t+1$ *honest affirmed* parties. Now if the $f'_i(0)$ values of the honest *affirmed* parties define a unique degree t polynomial, say $F'(x)$, then the unique defined secret $s^* = F'(0)$. Otherwise s^* is NULL. □

Lemma 3.54 (Weak Commitment) *Protocol* 1-Round-WSS *satisfies weak commitment property.*

PROOF: We have to consider the case when D is corrupted. We first prove that if D is corrupted, then he can not define two different set of *affirmed* parties, say C_1 and C_2 (each of size at least $2t+1$), defining two different secrets, say s_1 and s_2, such that in the reconstruction phase, depending upon the behavior of the corrupted parties, he can force reconstruction of either s_1 or s_2. In other words, in reconstruction phase if some set, say C of at least $2t+1$ *affirmed* parties are obtained, then D must have *uniquely* fixed (defined) it during sharing phase. The proof goes as follows: Assume that D had defined two different set of *affirmed* parties, C_1 and C_2, each of size at least $2t+1$. By this we mean that the polynomials and the theirs values are distributed properly among the parties in C_1 (and C_2 separately). Now each of the above sets contains at least $t+1$ honest parties. Since $n = 3t+1$, C_1 and C_2 must have $t+1$ parties in common. Let $\mathcal{H}_{com}$ denote the set of common honest parties in C_1 and C_2. Notice that $|\mathcal{H}_{com}| < t+1$ should hold to ensure that C_1 and C_2 define two distinct secrets. Now assume that during reconstruction phase, the corrupted D, along with the remaining $t-1$ corrupted parties, wants to force the reconstruction of the secret defined by C_1. We show that this is impossible and $NULL$ will be reconstructed. The reason is that in this case, every honest party P_i in $C_2 \setminus \mathcal{H}_{com}$ will be *semi-affirmed*, as P_i will be accepted by all the honest parties (at least $t+1$) in C_2. Similarly, if the corrupted D, along with the remaining $t-1$ corrupted parties, wants to force the reconstruction of the secret defined by C_2, then again it will lead to the reconstruction of $NULL$. The reason is that in this case, every honest party P_i in $C_1 \setminus \mathcal{H}_{com}$ will be *semi-affirmed*, as P_i will be accepted by all the honest parties (at least $t+1$) in C_1. This proves our claim that if some set, say C of at least $2t+1$ *affirmed* parties is obtained in reconstruction phase, then D must have *uniquely* defined (fixed) it during sharing phase.

Once the uniqueness of C is proved, we next proceed to show that either the secret $s^* \in \mathbb{F} \cup \{NULL\}$ defined by honest parties in C (see Claim 3.53) or $NULL$ will be reconstructed. If $s^* = NULL$, then irrespective of the $f'_i(0)$ corresponding to corrupted $P_i \in C$, $NULL$ will be reconstructed. But if $s^* \in \mathbb{F}$, then depending upon the $f'_i(0)$ corresponding to corrupted $P_i \in C$, either s^* or $NULL$ will be reconstructed. □

Theorem 3.55 *There exists an efficient 1-round sharing, 2-round reconstruction $(3t+1, t)$ statistical WSS protocol.*

PROOF: Protocol 1-Round-WSS presented here achieves correctness, except with error probability ϵ and also achieves weak commitment and secrecy without any error. This follows from Lemma 3.50, 3.52 and 3.54. □

3.9.1 1-round Sharing WSS with One Round of Reconstruction

It is interesting to note that if we restrict the adversary to a non-rushing adversary then the two rounds of the reconstruction phase can be collapsed into a single round. The two rounds are needed in order to force the adversary to commit to the polynomials $f_i(x)$ of the faulty parties prior to seeing the evaluation points, as this knowledge can enable the adversary to publish an incorrect polynomial that is accepted by the honest parties, which would violate the correctness of the protocol. However, if the adversary is non-rushing then this property is achieved via the synchronicity of the step. Therefore, we have the following theorem:

Theorem 3.56 *If the adversary $\mathcal{A}_t$ is non-rushing then there exists an efficient 1-round sharing 1-round reconstruction $(3t + 1, t)$ statistical WSS protocol.*

3.10 Efficient 3-round Sharing, 2-round Reconstruction $(2t + 1, t)$ Statistical WSS

Here we design a 3-round sharing, 2-round reconstruction $(2t + 1, t)$ statistical WSS protocol called 3-Round-WSS. In perfect settings, any three or more round sharing WSS protocol requires at least $3t + 1$ parties [73]. Therefore, we see that even in the case of WSS, probabilistically relaxing the conditions helps to improve fault tolerance.

The Intuition: To share a secret s, D chooses a degree t polynomial $f(x)$ with $f(0) = s$ and delivers his IC signature on $f(i)$ to party P_i. This prevents a corrupted party from producing incorrect share during reconstruction phase when D is honest. Hence it ensures **correctness** property of WSS. However notice that if D is corrupted, then a corrupted P_i can forge D's IC signature on any value and produce it during reconstruction phase. Even then, the protocol will satisfy the **weak commitment** property. The protocol is now given in Fig. 3.8.

Our protocol has an error probability of ϵ. To bound the error probability by ϵ, the computation in our statistical WSS protocol is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^2 2^{-\kappa}$. This is derived from the fact that in our WSS protocol, MVMS-ICP will be invoked with $\frac{\epsilon}{n}$ error probability and as mentioned in Chapter 2, $\epsilon \geq n2^{-\kappa}$ should hold to bound error probability of MVMS-ICP by ϵ.

Lemma 3.57 (Secrecy) *Protocol 3-Round-WSS satisfies secrecy property.*

PROOF: Easy. Follows from Lemma 2.6 and the fact that $f(x)$ is a t degree polynomial and $\mathcal{A}_t$ has only t points on it. □

Lemma 3.58 (Correctness) *Protocol 3-Round-WSS satisfies correctness property, except with probability ϵ.*

PROOF: We have to consider the case when D is honest. It is easy to see that if D is honest then all the honest P_i's (at least $t+1$) will always be present in REC. For every honest $P_i \in REC$, the revealed s_i will be equal to $f(i)$ without any error. Even for a corrupted $P_i \in REC$, the same will hold, except with probability $\frac{\epsilon}{n}$ (by **ICP-Correctness3**). Now there can be at most t corrupted parties in REC. Thus, except with probability $t\frac{\epsilon}{n} \approx \epsilon$, s_i's corresponding to all corrupted parties in REC will be equal to $f(i)$. This implies that s_i's corresponding to all the parties in REC will lie on degree t polynomial $f(x)$ and hence $s = f(0)$ will be reconstructed by each honest party, except with probability ϵ. □

Lemma 3.59 (Weak Commitment) *Protocol 3-Round-WSS satisfies weak commitment property, except with probability ϵ.*

Figure 3.8: A 3-Round Sharing 2-Round Reconstruction $(2t + 1, t)$ Statistical WSS

3-Round-WSS$(D, \mathcal{P}, s, \epsilon)$

Sharing Phase: Three Rounds

Round 1: D chooses a random degree t polynomial $f(x)$ such that $f(0) = s$. For $i = 1, \ldots, n$, D passes $ICSig(D, P_i, \mathcal{P}, s_i)$ having $\frac{\epsilon}{n}$ error to party P_i, where $s_i = f(i)$.

Round 3: P_i receives $ICSig(D, P_i, \mathcal{P}, s_i)$ having $\frac{\epsilon}{n}$ error.

Reconstruction Phase: Two Rounds

Round 1 and 2: For $i = 1, \ldots, n$, party P_i reveals $ICSig(D, P_i, \mathcal{P}, s_i)$ having $\frac{\epsilon}{n}$ error.

Local Computation (By Each Party):

1. Let REC be the set of all P_i's, such that P_i is successful in revealing $ICSig(D, P_i, \mathcal{P}, s_i)$. If $|REC| < t + 1$, then output $NULL$.
2. Check whether s_i values corresponding to the parties in REC define a unique degree t polynomial, say $f(x)$. If yes then output $s = f(0)$ as the secret. Otherwise output $NULL$.

PROOF: We have to consider the case when D is corrupted. Let $\mathcal{H}$ denote the set of honest parties in $\mathcal{P}$. As there are at least $t+1$ parties in $\mathcal{H}$, let $\overline{f}(x)$ be the polynomial defined by the s_i values held by the parties in $\mathcal{H}$ (on which they hold IC signature of D). If $\overline{f}(x)$ is of degree t, then we define D's committed secret as $\overline{s} = \overline{f}(0)$. Otherwise, we say that D has committed $\overline{s} = NULL$. We now show that in the reconstruction phase either $\overline{s}$ or $NULL$ will be reconstructed, except with probability ϵ.

We first claim that a party in $\mathcal{H}$ will be present in REC, except with probability $\frac{\epsilon}{n}$. This follows from **ICP-Correctness2**, according to which each honest P_i will be successful in revealing $ICSig(D, P_i, \mathcal{P}, s_i)$, except with probability $\frac{\epsilon}{n}$. Since $|\mathcal{H}| \geq t + 1$, all the honest parties will be present in REC, except with probability ϵ. However, a corrupted $P_i \in \mathcal{P}$ may be successful in revealing $ICSig(D, P_i, \mathcal{P}, \overline{s_i})$ for any $\overline{s_i}$ (as D is corrupted here) and can be included in REC.

Now if D's committed secret $\overline{s} = \overline{f}(0)$, then depending on the values revealed by corrupted parties in REC, either $\overline{s}$ or $NULL$ will be reconstructed. However, if D's committed secret $\overline{s} = NULL$, then irrespective of the values revealed by the corrupted parties in REC, $NULL$ will be reconstructed. Both the above happens, except with probability ϵ. Hence the lemma. □

Theorem 3.60 *There exists an efficient 3-round sharing, 2-round reconstruction $(2t + 1, t)$ statistical WSS protocol.*

PROOF: Protocol 3-Round-WSS presented here achieves correctness and weak commitment except with error probability ϵ and also achieves perfect secrecy. This follows from Lemma 3.57, 3.58 and 3.59. □

3.10.1 3-round Sharing WSS with One Round of Reconstruction

As in our VSS protocols, if we restrict the adversary to a non-rushing adversary then the two rounds of reconstruction phase of protocol 3-Round-WSS can be collapsed into a single round. Hence, we have the following theorem:

Theorem 3.61 *If the adversary $\mathcal{A}_t$ is non-rushing then there exists an efficient 3-round sharing 1-round reconstruction $(2t+1, t)$ statistical WSS protocol.*

3.11 Lower Bounds for Statistical WSS

We now prove the optimality of our 1-round and 2-round sharing $(3t + 1, t)$ statistical WSS protocol, with respect to the resiliency.

Theorem 3.62 *There is no 1-round and 2-round sharing (n, t) statistical WSS protocol with $n \leq 3t$, irrespective of the number of rounds in the reconstruction phase.*

PROOF: The proof for 2-round sharing (n, t) statistical WSS is provided in Theorem 3.35. Now it is obvious that 1-round sharing (n, t) statistical WSS protocol with $n \leq 3t$ will be impossible irrespective of the number of rounds in the reconstruction phase. □

We have proved the tightness of the above theorem by providing 1-round sharing $(3t + 1, t)$ statistical WSS protocol in section 3.9 and 2-round sharing $(3t + 1, t)$ statistical WSS in section 3.3.

We now prove the optimality of our 3-round sharing $(2t+1, t)$ statistical WSS protocol, with respect to the sharing rounds. It is well known that there does not exist any (n, t) VSS (hence WSS) with $n < 2t$ for any number of sharing as well as reconstruction rounds. So the best we can hope for is $(2t + 1, t)$ WSS protocol. Now since Theorem 3.62 says that for 1-round as well as for 2-round sharing WSS, $3t + 1$ is minimum, it automatically follows that 3-round sharing is optimal for $2t + 1$.

3.12 Conclusion and Open Problems

We obtain the following insightful conclusions from this chapter: (a) Existing lower bounds of perfect VSS and WSS can be circumvented by incorporating negligible error probability; (b) Probabilistically relaxing the conditions of VSS and WSS helps to increase the fault tolerance. This chapter leaves several interesting open problems:

Open Problem 3 *What is the lower bound on the total number of rounds in VSS, i.e. sharing plus reconstruction?*

This problem is also closely connected to the following question:

Open Problem 4 *Is it possible to design a 2-round statistical VSS protocol which satisfies the strong definition of statistical VSS mentioned in Definition 3.4?*

If the above question is answered in affirmative, then it would immediately result in a total of 3-round $(3t+1, t)$ statistical VSS protocol, as now the reconstruction can be achieved in one round with the help of error correction. Another interesting open problem is:

Open Problem 5 *Does allowing negligible error probability in* **Secrecy** *property of VSS (and WSS) brings any change in the round complexity of VSS and WSS presented in this chapter?*

Recently, [112] has reported an exponential 3-round sharing (and 2-round reconstruction) $(2t+1, t)$ statistical VSS and a polynomial 4-round sharing (and 2-round reconstruction) $(2t+1, t)$ statistical VSS. Therefore, another left-out open problem is:

Open Problem 6 *Does there exist a polynomial 3-round sharing $(2t+1, t)$ statistical VSS with $t > 1$?*

Chapter 4

Communication and Round Efficient Statistical VSS

In the previous chapter, we were concerned on the round complexity of statistical VSS and WSS protocols and therefore communication complexity was not given much importance. In this chapter, we concentrate on designing statistical VSS protocol that is simultaneously communication efficient as well as round efficient. Specifically, here we design statistical VSS with optimal resilience i.e with $n = 2t + 1$ parties (plus a broadcast channel is available) that achieves the best known communication and round complexity in the literature. Our VSS uses the ICP presented in Chapter 2 as the vital black box primitive. Though our VSS is of independent interest, we use it to propose a new and *robust* multiplication protocol for generating multiplication triples (that will be used in our MPC protocol) in the next chapter.

4.1 Introduction

4.1.1 Relevant Literature on Statistical VSS

VSS is a fundamental primitive used in many secure distributed computing protocols including MPC. Statistical VSS assuming $n = 2t+1$ parties and a common broadcast channel was first reported in [138, 137]. Later more efficient statistical VSS protocols with $n = 2t + 1$ are proposed in [48] and [49].

4.1.2 Our Network and Adversary Model

The network and adversary model is same as the one presented in in Section 2.1.2 of Chapter 2. Recall that the set of parties is denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$ and t out of the n parties can be under the influence of a *computationally unbounded powerful, Byzantine (active), rushing adversary*, denoted as $\mathcal{A}_t$. Apart from pairwise secure channels, there is a physical broadcast channel available in the network. In this chapter, we assume $n = 2t + 1$.

4.1.3 Contribution of This Chapter

Round complexity and communication complexity are the two important complexity measures of any fault-tolerant distributed computing protocol such as VSS. In this chapter, we look into both the complexity measures of statistical

VSS with optimal resilience and present a protocol that provides the best known communication and round complexity so far in the literate. Our protocol can deal with multiple secrets concurrently and thus can harness many advantages offered by dealing with multiple secrets simultaneously. Thus the communication complexity of our protocol for sharing multiple secrets simultaneously are better than multiple executions of protocols for sharing single secret. Our statistical VSS protocol satisfies the strong definition of statistical VSS (see Definition 3.4).

We now compare our VSS with the existing VSS of [138, 48, 49] and show its superiority. All the three schemes of [138, 48, 49] are designed for single secret. But they can be extended to deal with ℓ secrets in a straight forward manner by ℓ parallel invocations of the protocols. In Table 4.1, we compare our VSS scheme with the existing statistical VSS schemes in terms of communication and round complexity, where ϵ denotes the error probability of the protocols.

Table 4.1: Communication Complexity and Round Complexity of our statistical VSS and Existing Statistical VSS Schemes with $n = 2t + 1$

	Communication Complexity in bits		Round Complexity		# Secret
Ref.	Sharing	Rec.	Sharing	Rec.	
[138]	Private & Broadcast– $\Omega(n^4(\log \frac{1}{\epsilon})^3)$	Broadcast– $\Omega(n^4(\log \frac{1}{\epsilon})^3)$	at least 8	2	1
[48]	Private & Broadcast– $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$	Broadcast– $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$	10	2	1
[49]	Private & Broadcast– $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$	Private– $\mathcal{O}(n^2 \log \frac{1}{\epsilon})$ [a]	at least 16	2 [b]	1
This chapter	Private & Broadcast– $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$	Broadcast– $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$	5	2	ℓ

[a] This communication complexity is shown to be optimal in [49].

[b] The round complexity of 2 for the reconstruction phase of $(2t + 1, t)$ statistical VSS is shown to optimal in [49] when the adversary is assumed to be *rushing* (which is what we have assumed in this work).

Our protocol uses the ICP presented in Chapter 2 as the main building block.

Though our VSS is of independent interest, we use it to propose a new and novel multiplication protocol with robust fault handling mechanism for generating multiplication triples (that will be used in our MPC protocol) in the next chapter.

Our statistical VSS protocol involves a negligible error probability of ϵ. To bound the error probability by ϵ, all computation in our protocol are performed over a finite field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^3 2^{-\kappa}$. This is derived from the fact that in our VSS protocol, MVMS-ICP (the ICP of Chapter 2) will be invoked with $\frac{\epsilon}{n^2}$ error probability and as mentioned in Chapter 2, $\epsilon \geq n2^{-\kappa}$ should hold to bound error probability of MVMS-ICP by ϵ. So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{n^3}{\epsilon}) = \mathcal{O}(3 \log n + \log \frac{1}{\epsilon}) = \mathcal{O}(\log \frac{1}{\epsilon})$ bits (the last equality in the sequence follows from relation $n = \mathcal{O}(\log \frac{1}{\epsilon})$).

In order to bound the error probability of our VSS protocol by some specific value of ϵ, we find out the minimum value of κ that satisfies $\epsilon \geq n^3 2^{-\kappa}$. This value for κ will consequently determine the field $\mathbb{F}$ over which our protocol should work.

4.1.4 The Road-map

For the sake of simplicity, we first present our VSS for a single secret (the main idea behind our protocol will be clear from this protocol) in section 4.2 and then extend it for multiple secrets in section 4.3. We conclude this chapter in section 4.4.

4.2 Statistical VSS For a Single Secret

In this section, we present a new statistical VSS protocol with $n = 2t+1$ parties that can share/commit a single secret. In the next section, we will further extend this protocol to obtain a VSS which can deal with multiple secrets concurrently. Our protocol follows strong definition of statistical VSS presented in Definition 3.4 and requires five rounds in the sharing phase and two rounds in the reconstruction phase.

The Intuition: The high level idea of the protocol is as follows: D selects a random symmetric bivariate polynomial $F(x, y)$ of degree t in x and y, such that $F(0, 0) = s$ and sends $f_i(x)$ to party P_i. At the end the sharing phase, if D is not discarded then every honest P_i holds a degree t polynomial $f_i(x)$ such that for every pair of honest parties (P_i, P_j), $f_i(j) = f_j(i)$. This implies that if D is not discarded, then the $f_i(x)$ polynomials of the honest parties define a symmetric bivariate polynomial $F(x, y)$. Moreover in the protocol, we ensure that no corrupted P_i will be able to disclose $\overline{f_i}(x) \neq f_i(x)$ in reconstruction phase, with very high probability. Hence irrespective of whether D is honest or corrupted, reconstruction of $s = F(0, 0)$ is enforced, except with negligible probability of ϵ. To achieve all the above properties, in our protocol, D gives his IC Signature to individual parties. Concurrently every individual party also gives his IC Signature to every other party. The protocol is somewhat inspired by the VSS protocol of [48]. The formal details of our protocol are given in Fig. 4.1 and Fig. 4.2.

We now prove the properties of our VSS scheme.

Claim 4.1 *An honest D will not be* **discarded** *in sharing phase, with probability at least* $(1 - \epsilon)$.

PROOF: If D is honest, then a pair of honest parties can never be a **conflicting pair**. Now from the conditions stated in step 1 of **Local Computation** of 5VSS-Share, it is clear that an honest D will be discarded if somehow any corrupted party P_i (there are at most t such parties) is able to reveal $ICSig(D, P_i, \mathcal{P}, \overline{f_i}(j))$ with $\overline{f_i}(j) \neq f_i(j)$ for any $j \in \{0, \ldots, n\}$. We show that this can happen only with probability at most ϵ.

By **ICP-Correctness3**, a corrupted P_i will be successful in revealing $ICSig(D, P_i, \mathcal{P}, \overline{f_i}(j))$ with $\overline{f_i}(j) \neq f_i(j)$, with probability $\epsilon' = \frac{\epsilon}{n^2}$ (recall that each IC signature has $\epsilon' = \frac{\epsilon}{n^2}$ error). As there are t corrupted parties and $n+1$ possible values for j, the event that some corrupted party will be able to reveal $ICSig(D, P_i, \mathcal{P}, \overline{f_i}(j))$ with $\overline{f_i}(j) \neq f_i(j)$ for some j may occur with probability at most $t(n+1)\epsilon' \approx \epsilon$. Hence the claim. □

Claim 4.2 *If D is not discarded in 5VSS-Share, then there exists a unique symmetric bivariate polynomial $F(x, y)$ of degree t in both x and y, such that $f_i(x)$*

Figure 4.1: Sharing Phase of 5-Round Sharing, 2-Round Reconstruction $(2t+1, t)$ Statistical VSS

5VSS$(D, \mathcal{P}, s, \epsilon)$

Sharing Phase — 5VSS-Share$(D, \mathcal{P}, s, \epsilon)$: This will take five rounds

Round 1:

1. D chooses a random symmetric bivariate polynomial $F(x, y)$ of degree t in both x and y, such that $F(0, 0) = s$ and sends $ICSig(D, P_i, \mathcal{P}, f_i(j))$ having $\epsilon' = \frac{\epsilon}{n^2}$ error to P_i, for every $j = 0, \ldots, n$, where $f_i(x) = F(x, i)$.
2. For $i = 1, \ldots, n$, party P_i selects a random $r_{ij} \in \mathbb{F}$ and sends $ICSig(P_i, P_j, \mathcal{P}, r_{ij})$ having $\epsilon' = \frac{\epsilon}{n^2}$ error to party P_j for all $j = 1, \ldots, n$.

Round 2:

1. Party P_i broadcasts: (a) $a_{ij} = f_i(j) + r_{ij}$, (b) $b_{ij} = f_i(j) + r_{ji}$ for $j = 1, \ldots, n$.

Round 3:

1. Party P_i receives $ICSig(D, P_i, \mathcal{P}, f_i(j))$ having ϵ' error from D for $j = 0, \ldots, n$.
2. Party P_i receives $ICSig(P_j, P_i, \mathcal{P}, r_{ji})$ having ϵ' error from P_j for $j = 1, \ldots, n$.

At the end of **Round 3**, a pair (P_i, P_j) is called as **conflicting pair** if one of the following holds:

- If $a_{ij} \neq b_{ji}$.
- P_i had broadcasted r_{ij} during **Round 2** of Ver$(P_i, P_j, \mathcal{P}, r_{ij}, \epsilon')$.
- D had broadcasted $f_i(j)$ during **Round 2** of Ver$(D, P_i, \mathcal{P}, f_i(j), \epsilon')$.

Round 4:

1. For every **conflicting pair** (P_i, P_j), party P_i reveals $ICSig(D, P_i, \mathcal{P}, f_i(j))$ and party P_j reveals $ICSig(D, P_j, \mathcal{P}, f_j(i))$, each having ϵ' error.
2. P_i reveals $ICSig(D, P_i, \mathcal{P}, f_i(j))$ for $j = 0, \ldots, n$, each having ϵ' error, if $(f_i(0), \ldots, f_i(n))$ do not define a degree t polynomial.
3. Both the above steps will continue in **Round 5** as well because revealing IC signature requires two rounds.

Local Computation at the end of Round 5 (By Every Party)

1. D will be **discarded** and the protocol will terminate here, if one of the following happens:
   - For a **conflicting pair** (P_i, P_j), both P_i and P_j are successful in revealing $ICSig(D, P_i, \mathcal{P}, f_i(j))$ and $ICSig(D, P_j, \mathcal{P}, f_j(i))$ respectively AND $f_i(j) \neq f_j(i)$.
   - Some P_i is successful in revealing $ICSig(D, P_i, \mathcal{P}, f_i(j))$ for every $j = 0, \ldots, n$ AND $(f_i(0), \ldots, f_i(n))$ revealed by P_i, do not define a degree t polynomial.
2. If D is not discarded, then every P_i computes $ICSig(P_j, P_i, \mathcal{P}, b_{ji} - r_{ji})$ (which is same as $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$), corresponding to every P_j, such that (P_i, P_j) is not a **conflicting pair**. Accordingly every party computes verification information corresponding to $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$. (Follows from Linearity of IC Signature presented in Section 2.5 of Chapter 2).

held by every honest P_i at the end of 5VSS-Share satisfies $F(x, i) = f_i(x)$ with probability at least $(1 - \epsilon)$.

Proof: Assuming that D is not discarded in 5VSS-Share, the above claim

Figure 4.2: Reconstruction Phase of 5-round sharing 2-round reconstruction $(2t+1, t)$ statistical VSS

5VSS($D, \mathcal{P}, s, \epsilon$**)**

Reconstruction Phase — 5VSS-Rec($D, \mathcal{P}, s, \epsilon$): This will take two rounds

Round 1 and 2: If D is not discarded during 5VSS-Share, then every P_i reveals $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$, having ϵ' error, such that (P_i, P_j) is not a **conflicting pair**.

Local Computation at the end of Round 2 of 5VSS-Rec (By Every Party)

1. Create a set REC. Add P_i to REC if:
   - P_i is successful in revealing $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$ for all P_j such that (P_i, P_j) is not a **conflicting pair AND**
   - $(f_{i1}, \ldots, f_{in})$ define a degree t polynomial, say $f_i(x)$ where:
     - f_{ij} is equal to $f_j(i)$ revealed by P_i in reconstruction phase when (P_i, P_j) is **not** a **conflicting pair**;
     - $f_{ij} = f_i(j)$ when (P_i, P_j) is a **conflicting pair** and P_i had successfully revealed $ICSig(D, P_i, \mathcal{P}, f_i(j))$ during 5VSS-Share.
2. Reconstruct a symmetric bivariate polynomial of degree t in x and y, say $F(x, y)$, such that $F(x, i) = f_i(x)$ for every $P_i \in REC$.
3. Output $s = F(0, 0)$.

follows if $f_i(x)$ held by every honest P_i is of degree t and for every honest pair (P_i, P_j), $f_i(j) = f_j(i)$ holds with probability at least $(1 - \epsilon)$. When D is honest then above statement holds without any error probability; i.e $\epsilon = 0$ when D is honest. So for the rest of the proof, we assume D to be corrupted.

We now show that an honest P_i will hold a degree t polynomial $f_i(x)$ with probability at least $(1 - \frac{\epsilon}{n})$. This will in turn assert that all the honest parties (there are at least $(t+1)$ honest parties) will hold degree t polynomials with probability at least $(1 - (t+1)\frac{\epsilon}{n}) \approx (1 - \epsilon)$. Let an honest P_i had received a polynomial $f_i(x)$ of degree more than t from D. So in this case, P_i will reveal $ICSig(D, P_i, \mathcal{P}, f_i(j))$ for every $j = 0, \ldots, n$. P_i can successfully reveal the above signatures with probability at least $(1 - \epsilon')^n \approx (1 - n\epsilon') \approx (1 - \frac{\epsilon}{n})$ and thus with probability $(1 - \frac{\epsilon}{n})$, P_i can prove that $(f_i(0), \ldots, f_i(n))$ do not define degree t polynomial. This will lead to discarding D in sharing phase which is a contradiction. Hence with probability $(1 - \frac{\epsilon}{n})$, an honest P_i holds a degree t polynomial $f_i(x)$.

Next we assert that for an honest pair (P_i, P_j), $f_i(j) = f_j(i)$ will hold with probability at least $(1 - \epsilon')$. We consider two cases:

1. (P_i, P_j) is **not** a **conflicting pair**: Here $f_i(j) = f_j(i)$ will hold without any error;
2. (P_i, P_j) is a **conflicting pair**: Here also $f_i(j) = f_j(i)$ will hold good with

very high probability, as otherwise both P_i and P_j would have successfully revealed $ICSig(D, P_i, f_i(j))$ and $ICSig(D, P_i, f_j(i))$ respectively (by **ICP-Correctness2**) with probability at least $(1-\epsilon')^2 \approx (1-2\epsilon') \approx (1-\epsilon')$ such that $f_i(j) \neq f_j(i)$ and thus D would have been discarded which is a contradiction.

As there are at least $(t+1)^2$ honest pairs, for all honest pairs (P_i, P_j), $f_i(j) = f_j(i)$ will hold with probability at least $(1-(t+1)^2\epsilon') \approx (1-\epsilon)$. Hence the claim. □

Remark 4.3 (D's Commitment in 5VSS-Share) *The polynomial $F(x,y)$ defined in Claim 4.2 is called D's committed bivariate polynomial in protocol 5VSS-Share. The value $s = F(0,0)$ is called D's commitment in 5VSS-Share.*

Claim 4.4 *In protocol 5VSS-Rec, for all $P_i \in REC$, polynomial $f_i(x)$ satisfying $F(x,i) = f_i(x)$ is reconstructed with probability at least $(1-\epsilon)$, where $F(x,y)$ is D's committed bivariate polynomial in 5VSS-Share.*

PROOF: To prove the lemma, we show that for a $P_i \in REC$, $f_i(x)$ satisfying $F(x,i) = f_i(x)$ is reconstructed with probability at least $(1-\frac{\epsilon}{n})$. This will assert that for all $P_i \in REC$, the above will hold with probability at least $(1 - |REC|\frac{\epsilon}{n}) \approx (1-\epsilon)$ (as $|REC| \geq t+1$). We now have two cases: (a) when P_i is honest and (b) when P_i is corrupted.

So first consider an honest $P_i \in REC$. From the proof of Claim 4.2, P_i holds $f_i(x) = F(x,i)$ in 5VSS-Share with probability at least $(1-\frac{\epsilon}{n})$. At the end of 5VSS-Share, P_i holds $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$ for every P_j such that (P_i, P_j) is **not** a *conflicting pair*. Now in 5VSS-Rec, by **ICP-Correctness2**, honest P_i will successfully reveal $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$ for a j (such that (P_i, P_j) is **not** a *conflicting pair*) with probability at least $(1-\epsilon')$. Also for every P_j such that (P_i, P_j) is a *conflicting pair*, P_i had successfully revealed $ICSig(P_j, P_i, \mathcal{P}, f_i(j))$ (without any error when D is honest, by **ICP-Correctness1**; with probability at least $(1-\epsilon')$ when D is corrupted, by **ICP-Correctness2**). Hence, in 5VSS-Rec, for honest P_i, $f_{ij} = f_j(i)$ for all j with probability at least $(1-n\epsilon') = (1-\frac{\epsilon}{n})$. Hence the lemma holds for an honest $P_i \in REC$, irrespective of whether D is honest or corrupted.

Now we show the lemma for a corrupted $P_i \in REC$. In 5VSS-Share, for every **honest** P_j such that (P_i, P_j) is **not** a *conflicting pair*, P_i had received $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$ from P_j. Now in 5VSS-Rec, P_i must have revealed $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$ for a j such that (P_i, P_j) is **not** a *conflicting pair* with probability at least $(1-\epsilon')$ (by **ICP-Correctness3**). Also for every **honest** P_j such that (P_i, P_j) is a *conflicting pair*, P_i had successfully revealed $ICSig(D, P_i, \mathcal{P}, f_i(j))$ (in 5VSS-Share) satisfying $f_i(j) = f_j(i)$ as otherwise D would have been discarded. Hence P_i has revealed $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$ for all j with probability at least $(1-n\epsilon') \approx (1-\frac{\epsilon}{n})$. This implies that in 5VSS-Rec for corrupted P_i, $f_{ij} = f_j(i) = f_i(j)$ for every **honest** P_j with probability at least $(1-\frac{\epsilon}{n})$. Now since there are at least $(t+1)$ honest parties and $(f_{i1}, \ldots, f_{in})$ define a degree t polynomial $f_i(x)$, it clearly implies that $f_i(x) = F(x,i)$ with probability $(1-\frac{\epsilon}{n})$. Hence the claim. □

Lemma 4.5 (Secrecy) *Protocol 5VSS satisfies perfect secrecy.*

PROOF: Here we have to consider D to be honest. Without loss of generality, let $P_1, \ldots, P_t$ be under the control of adversary. So adversary will learn

$f_1(x), \ldots, f_t(x)$. Now from **ICP-Secrecy**, for every pair of honest parties (P_i, P_j), the values r_{ij} and r_{ji} will be unknown to the adversary. Hence a_{ij}, b_{ij} broadcasted by P_i and a_{ji}, b_{ji} broadcasted by P_j do not reveal any information on $f_i(j) = f_j(i)$. Moreover, honest D will never broadcast $f_i(j)$ or $f_j(i)$ for honest P_i and P_j. So from the properties of symmetric bivariate polynomial of degree t in x and y, adversary will fall short by one point for uniquely reconstructing $F(x, y)$ and hence $s = F(0, 0)$ will remain information theoretically secure. □

Lemma 4.6 (Correctness) *Protocol 5VSS satisfies correctness property with probability at least* $(1 - \epsilon)$.

PROOF: Here we have to consider the case when D is honest. By Claim 4.1, honest D will never be discarded in sharing phase, except with probability ϵ. Now by Claim 4.2, D will commit polynomial $F(x, y)$ and by Claim 4.4 for all $P_i \in REC$, $f_i(x) = F(x, i)$ will be reconstructed with probability at least $(1 - \epsilon)$. Moreover $|REC|$ will be at least $t + 1$ as it will contain at least the honest parties. So using $f_i(x)$ of the parties in REC, $F(x, y)$ will be reconstructed with probability at least $(1 - \epsilon)$. □

Lemma 4.7 (Strong Commitment) *Protocol 5VSS satisfies strong commitment property with probability at least* $(1 - \epsilon)$.

PROOF: Here we have to consider the case when D is corrupted. If D is **discarded** during sharing phase then strong commitment holds trivially, as every party may assume some predefined default value $s^\star$ as D's commitment. On the other hand, when D is not discarded, the proof follows from the same argument as given in Lemma 4.6. □

Theorem 4.8 *Protocol 5VSS is an efficient* $(2t + 1, t)$ *statistical VSS protocol.*

PROOF: This follows from Lemma 4.5, 4.6 and 4.7. □

Theorem 4.9 *In protocol 5VSS, the sharing phase protocol 5VSS-Share requires 5 rounds and the reconstruction phase protocol 5VSS-Rec requires 2 rounds.*

PROOF: Follows from the protocol steps presented in Fig. 4.1 and 4.2. □

Theorem 4.10 *Protocol 5VSS achieves the following communication complexity bounds:*

- *Protocol 5VSS-Share requires both private as well as broadcast communication of* $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ *bits.*
- *Protocol 5VSS-Rec requires broadcast communication of* $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ *bits.*

PROOF: The communication complexity of 5VSS-Share follows from the fact that there can be at most $\mathcal{O}(n^2)$ executions of Gen and Ver with $\ell = 1$. Similarly, the communication complexity of 5VSS-Rec follows from the fact that there can be at most $\mathcal{O}(n^2)$ executions of Reveal with $\ell = 1$ value. □

The next two subsections are important for the next chapter where we use our VSS as building block in our MPC.

4.2.1 The Output Generated by 5VSS-Share

At a glance the situation created at the end of 5VSS-Share is as follows (if D is not **discarded**): There is some symmetric bivariate polynomial $F(x, y)$ such that every *honest* party P_i holds polynomial $f_i(x) = F(x, i)$ and every P_j holds an IC signature on $f_j(i)$ from P_i as well as from D when (P_i, P_j) is not a **conflicting pair**. For every other i such that (P_i, P_j) is a **conflicting pair**, the value $f_j(i)$ is available publicly. For the ease of reference, we use the following definitions to capture the output of 5VSS-Share:

Definition 4.11 ($1d^\star$-sharing) *We say that a party $P \in \mathcal{P}$ has $1d^\star$-shared (here $1d$ stands for one-dimensional) a secret $s \in \mathbb{F}$ among the parties in $\mathcal{P}$, if the following holds:*

1. *There exists degree t polynomial $f(x)$ with $f(0) = s$;*
2. *The i^{th} value on $f(x)$, namely $s_i = f(i)$, also called as i^{th} share of s, is either publicly known or otherwise party $P_i \in \mathcal{P}$ holds $ICSig(P, P_i, \mathcal{P}, s_i)$;*
3. *There can be at most t publicly known s_i values.*

The $1d^\star$-sharing of s is denoted by $\langle s \rangle_t$. If some specific party P does the sharing, then we denote it by $\langle s \rangle_t^P$.

Now note that at the end of 5VSS-Share, every honest P_i has done $1d^\star$-sharing of value $f_i(0)$ using polynomial $f_i(x)$. Additionally, D has done $1d^\star$-sharing of value $f_i(0)$ using the same polynomial where $f_i(j)$ is public for every P_j such that (P_i, P_j) is a conflicting pair (otherwise P_j holds $ICSig(D, P_j, \mathcal{P}, f_i(j))$ and $ICSig(P_i, P_j, \mathcal{P}, f_i(j))$). Now we introduce the definition of $2d^\star$-sharing.

Definition 4.12 ($2d^\star$-sharing) *A value $s \in \mathbb{F}$ is $2d^\star$-shared (here $2d$ stands for two-dimensional) among the parties in $\mathcal{P}$, denoted as $\langle\langle s \rangle\rangle_t$, if the following holds:*

1. *There exists degree t polynomials $f(x), f_1(x), \ldots, f_n(x)$, with $f(0) = s$;*
2. *For $i = 1, \ldots, n$, $f_i(0) = f(i)$;*
3. *Every honest party $P_i \in \mathcal{P}$ holds a share $s_i = f(i)$ of s and the polynomial $f_i(x)$. Moreover, P_i has $1d^\star$-shared the value s_i using polynomial $f_i(x)$.*

If some specific party P does the sharing, then we denote it by $\langle\langle s \rangle\rangle_t^P$.

We can easily see that at the end of 5VSS-Share, the secret $s = F(0, 0)$ is $2d^\star$-shared by D where the $f(x)$ in above definition is nothing but $F(x, 0)$ and $f_i(x) = F(x, i)$. We note that a secret s can be reconstructed robustly from its $2d^\star$-sharing using reconstruction phase protocol 5VSS-Rec. In protocol 5VSS-Share, D also does $1d^\star$-sharing of values $f_i(0)$, for $i = 0, \ldots, n$, which is not captured in the definition of of $2d^\star$-sharing. In fact these sharings are not required in 5VSS-Rec for the reconstruction of the secret. But we will use the $1d^\star$-sharing done by D in our multiplication protocol in order to either reconstruct $f_i(0)$ or detect that D is faulty. This follows from the fact that a secret may not be robustly reconstructed from its $1d^\star$-sharing. But if the reconstruction fails then it can be concluded that the party P who has done the $1d^\star$-sharing is corrupted.

We will elaborate on this in the next chapter. Robust reconstruction of secret is one major factor that differentiate between $1d^\star$-sharing and $2d^\star$-sharing. There is another difference (between these two sharings) which will be pointed out in the sequel.

There are $\Theta(n)$ and $\Theta(n^2)$ underlying IC signatures in $1d^\star$-sharing and $2d^\star$-sharing respectively. We now introduce the following definitions:

Definition 4.13 ($1d^\star$-sharing with ϵ Error) *We say that a $1d^\star$-sharing has ϵ error, if each of its $\Theta(n)$ underlying IC signature has $\frac{\epsilon}{n}$ error.*

In Section 5.3.1 (of next chapter), we will show that if the $1d^\star$-sharing of a secret s has ϵ error, then s can be reconstructed from its $1d^\star$-sharing, except with error probability ϵ when the party P who has done the sharing is honest. Additionally, if the secret is not reconstructed, then party P is corrupted except with probability ϵ.

Definition 4.14 ($2d^\star$-sharing with ϵ Error) *We say that a $2d^\star$-sharing has ϵ error, if each of its $\Theta(n^2)$ underlying IC signature has $\frac{\epsilon}{n^2}$ error.*

If a secret is $2d^\star$-shared using protocol 5VSS-Share, executed with error probability ϵ, then the resultant $2d^\star$-sharing will have ϵ error. From the proof of **Correctness** and **Strong Commitment** properties of 5VSS (see Lemma 4.6 and 4.7), if the $2d^\star$-sharing of a secret s has ϵ error, then s can be reconstructed from its $2d^\star$-sharing, except with probability ϵ, using protocol 5VSS-Rec. So we have the following important theorem:

Theorem 4.15 *Let $\langle\langle s\rangle\rangle_t$ be a $2d^\star$-sharing, having ϵ error. Then s can be correctly reconstructed from $\langle\langle s\rangle\rangle_t$, except with error probability ϵ.*

4.2.2 Linearity Property of $1d^\star$-sharing and $2d^\star$-sharing

Linearity Property of $1d^\star$-sharing: The $1d^\star$-sharing satisfies linearity property. Specifically, let $P \in \mathcal{P}$ be a party, who has done $1d^\star$-sharing of q secrets, say $s^1, \ldots, s^q$. Moreover, let the following holds, which we call as *condition for linearity of $1d^\star$-sharing: For $i = 1, \ldots, n$, the IC signatures $ICSig(P, P_i, \mathcal{P}, s_i^1), \ldots, ICSig(P, P_i, \mathcal{P}, s_i^q)$ (possibly some of s_i^j's are public) satisfy the condition of linearity of IC signatures (recall from subsection 2.5 of Chapter 2), where $s_i^1, \ldots, s_i^q$ denotes the i^{th} share of $s^1, \ldots, s^q$ respectively.* Then the parties can compute $1d^\star$-sharing of $s = s^1 + \ldots + s^q$ without doing any further communication. This is achieved by asking each party P_i to compute $ICSig(P, P_i, \mathcal{P}, s_i^1 + \ldots + s_i^q)$ from $ICSig(P, P_i, \mathcal{P}, s_i^1), \ldots, ICSig(P, P_i, \mathcal{P}, s_i^q)$ (this is possible as the signatures satisfy the *condition of linearity of IC signatures*), where $s_i^1 + \ldots + s_i^q$ denotes the i^{th} share of s. We capture this scenario by writing $\langle s\rangle_t^P = \sum_{i=1}^q \langle s_i\rangle_t^P$.

Notice that linearity property does not apply on $1d^\star$-sharing when the sharings are generated by different parties. That is, given $\langle s^1\rangle_t^P$ and $\langle s^2\rangle_t^Q$, where P and Q are two different parties, then the parties cannot locally compute $1d^\star$-sharing of $s^1 + s^2$. This is because the underlying IC signatures in $\langle s^1\rangle_t^P$ and $\langle s^2\rangle_t^Q$ will not satisfy the condition of linearity of IC signatures, as stated in Note 2.14 at the end of Section 2.5 of Chapter 2.

Linearity Property of $2d^\star$-sharing: Now similar to the linearity property of $1d^\star$-sharing, $2d^\star$-sharing also satisfies linearity property. Specifically, let $P \in \mathcal{P}$

be a party, who has done $2d^\star$-sharing of a number of secrets, say $s^1, \ldots, s^q$. Moreover, let the following holds, which we call as *condition for linearity of $2d^\star$-sharing*: *For every honest P_i, the $1d^\star$-sharing $\langle s_i^1 \rangle_t^{P_i}, \ldots, \langle s_i^q \rangle_t^{P_i}$ satisfies the condition for linearity of $1d^\star$-sharing. Here $s_i^1, \ldots, s_i^q$ denotes the i^{th} share of $s^1, \ldots, s^q$ respectively.* Then the parties can compute $2d^\star$-sharing of $s = s^1 + \ldots + s^q$ without doing any further communication. This is achieved by asking the parties to locally compute $\langle s_i^1 + \ldots + s_i^q \rangle_t^{P_i}$ from $\langle s_i^1 \rangle_t^{P_i}, \ldots, \langle s_i^q \rangle_t^{P_i}$ for all i, where $s_i^1 + \ldots + s_i^q$ denotes the i^{th} share of $s = s^1 + \ldots + s^q$. We capture this scenario by writing $\langle\langle s \rangle\rangle_t^P = \sum_{i=1}^q \langle\langle s^i \rangle\rangle_t^P$.

Notice that unlike $1d^\star$-sharing, we can apply linearity property on $2d^\star$-sharing even when the sharings are generated by different parties, provided the underlying $1d^\star$-sharing satisfies the *condition for linearity of $1d^\star$-sharing*. More specifically, let s^1 and s^2 be two values which are $2d^\star$-shared by two different parties, say P and Q; i.e., $\langle\langle s^1 \rangle\rangle_t^P$ and $\langle\langle s^2 \rangle\rangle_t^P$ are given. Moreover, for every honest P_i, let the underlying $1d^\star$-sharing $\langle s_i^1 \rangle_t^{P_i}$ and $\langle s_i^2 \rangle_t^{P_i}$ satisfies the condition for linearity of $1d^\star$-sharing. Then the parties can compute $2d^\star$-sharing of $s = s^1 + s^2$ without doing any further communication. This follows from the fact that the parties can locally compute $\langle s_i^1 + s_i^2 \rangle_t^{P_i}$ from $\langle s_i^1 \rangle_t^{P_i}$ and $\langle s_i^2 \rangle_t^{P_i}$, where $s_i^1 + s_i^2$ denotes the i^{th} share of $s = s^1 + s^2$. We capture this scenario by writing $\langle\langle s \rangle\rangle_t = \langle\langle s^1 \rangle\rangle_t^P + \langle\langle s^2 \rangle\rangle_t^Q$.

Before ending this section, we show that a linearly combined $1d^\star$-sharing/$2d^\star$-sharing will have ϵ error when each of the individual $1d^\star$-sharing/$2d^\star$-sharing has ϵ error.

Lemma 4.16 *Assume that $\langle s^1 \rangle_t^P, \ldots, \langle s^q \rangle_t^P$ are q different $1d^\star$-sharing, each having ϵ error. Let $\langle s \rangle_t^P = \sum_{j=1}^q \langle s^j \rangle_t^P$. Then $\langle s \rangle_t^P$ will have ϵ error.*

PROOF: Since each of $\langle s^1 \rangle_t^P, \ldots, \langle s^q \rangle_t^P$ has ϵ error, it implies that for every $i = 1, \ldots, n$, each IC signature $ICSig(P, P_i, \mathcal{P}, s_i^1), \ldots, ICSig(P, P_i, \mathcal{P}, s_i^q)$ will have $\frac{\epsilon}{n}$ error. This implies that $ICSig(P, P_i, \mathcal{P}, s_i^1 + \ldots + s_i^q)$ will have $\frac{\epsilon}{n}$ error (see Lemma 2.13). Now since $\langle s \rangle_t^P = \langle s^1 \rangle_t^P + \ldots + \langle s^q \rangle_t^P$, it follows that $\langle s \rangle_t^P$ will have ϵ error. □

Now using similar argument as above, we can prove the following lemma:

Lemma 4.17 *Assume that $\langle\langle s^1 \rangle\rangle_t, \ldots, \langle\langle s^q \rangle\rangle_t$ are q different $2d^\star$-sharing, each having ϵ error. Let $\langle\langle s \rangle\rangle_t = \sum_{j=1}^q \langle\langle s^j \rangle\rangle_t$. Then $\langle\langle s \rangle\rangle_t$ will have ϵ error.*

The above lemma implies that the individual s^j's as well as the sum value s can be reconstructed from their corresponding $2d^\star$-sharing, except error probability ϵ.

In the next chapter, when we use linearity property of $1d^\star$-sharing on a number of $1d^\star$-sharings, we assume that the *condition for linearity of $1d^\star$-sharing* has been satisfied. The same applies in case of $2d^\star$-sharing as well.

4.3 Statistical VSS For Multiple Secrets

In this section, we present a statistical VSS protocol with $n = 2t + 1$ parties that can share/commit ℓ secrets concurrently. This is the extension of the VSS protocol presented in the previous section. Again our protocol follows strong definition of statistical VSS presented in Definition 3.4.

We call our statistical VSS scheme as 5VSS-MS (here MS stands for multiple secrets) that allows to share a secret $S = (s^1, \ldots, s^\ell)$, containing ℓ elements from $\mathbb{F}$. Protocol 5VSS-MS consists of two sub-protocols, namely 5VSS-MS-Share (protocol corresponding to sharing phase) and 5VSS-MS-Rec (protocol corresponding to reconstruction phase). 5VSS-MS is a simple extension of 5VSS, presented in previous section. While using ℓ executions of 5VSS-Share, one for each $s^l \in S$, D can share S with a private and broadcast communication of $\mathcal{O}(\ell n^3 \log \frac{1}{\epsilon})$ bits, protocol 5VSS-MS-Share achieves the same task with a private and broadcast communication of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits. This shows that executing a *single instance* of 5VSS-MS dealing with *multiple secrets* concurrently is advantageous over executing *multiple instances* of 5VSS dealing with *single secret*. Protocol 5VSS-MS is presented in Fig. 4.3 and Fig. 4.4.

Since protocol 5VSS-MS is simple extension of protocol 5VSS, the proofs for the properties of 5VSS-MS will follow from the proofs of 5VSS. Hence to avoid repetitions, we do not present them again. Instead, we just state the following theorems.

Theorem 4.18 *5VSS-MS is an efficient $(2t+1, t)$ statistical VSS scheme for dealing with ℓ secrets concurrently.*

Theorem 4.19 *In 5VSS-MS, the sharing phase protocol 5VSS-MS-Share requires 5 rounds and the reconstruction phase protocol 5VSS-MS-Rec requires 2 rounds.*

Theorem 4.20 *Protocol 5VSS-MS achieves the following communication complexity bounds:*

- *Protocol 5VSS-MS-Share requires private as well as broadcast communication of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits.*
- *Protocol 5VSS-Rec requires broadcast communication of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits.*

PROOF: The communication complexity of 5VSS-MS-Share follows from the fact that it requires $\mathcal{O}(n^2)$ executions of Gen and Ver, dealing with ℓ values. Similarly, the communication complexity of 5VSS-MS-Rec follows from the fact that it requires $\mathcal{O}(n^2)$ executions of Reveal. □

The next two subsections are important for the next chapter where we use our VSS as building block in our MPC.

4.3.1 The Output Generated by 5VSS-MS-Share

Now similar to the way we have interpreted the output of 5VSS-Share, the output of 5VSS-MS-Share can also be captured by the following definitions which are in some sense extension of the definition of $1d^\star$-sharing and $2d^\star$-sharing respectively.

Definition 4.21 ($1d^{(\star,\ell)}$-sharing) *We say that a party P has $1d^{(\star,\ell)}$-shared $S = (s^1, \ldots, s^\ell) \in \mathbb{F}^\ell$ among the parties in $\mathcal{P}$, if the following holds:*

1. *There exists degree t polynomials $f^1(x), \ldots, f^\ell(x)$ with $f^l(0) = s^l$, for $l = 1, \ldots, \ell$;*

Figure 4.3: Sharing Phase of $(2t+1, t)$ statistical VSS Scheme 5VSS-MS

5VSS-MS-Share($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \epsilon$)

Round 1:

1. For $l = 1, \ldots, \ell$, D chooses a random symmetric bivariate polynomial $F^l(x, y)$ of degree t in both x and y, such that $F^l(0, 0) = s^l$ and sends $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ having $\epsilon' = \frac{\epsilon}{n^2}$ error to P_i for every $j = 0, \ldots, n$, where $f_i^l(x) = F^l(x, i)$.
2. For every P_j, party P_i selects ℓ random values $r_{ij}^1, \ldots, r_{ij}^\ell$ and sends $ICSig(P_i, P_j, \mathcal{P}, (r_{ij}^1, \ldots, r_{ij}^\ell))$ having ϵ' error to P_j.

Round 2:

1. For $l = 1, \ldots, \ell$, party P_i broadcasts: (a) $a_{ij}^l = f_i^l(j) + r_{ij}^l$, (b) $b_{ij}^l = f_i^l(j) + r_{ji}^l$.

Round 3:

1. Party P_i receives $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ having ϵ' error for every $j = 0, \ldots, n$ from D.
2. Party P_i receives $ICSig(P_j, P_i, \mathcal{P}, (r_{ji}^1, \ldots, r_{ji}^\ell))$ having ϵ' error from P_j, for $j = 1, \ldots, n$.

A pair (P_i, P_j) is called as **conflicting pair** if one of the following holds:

- If $a_{ij}^l \neq b_{ji}^l$ for some $l \in \{1, \ldots, \ell\}$.
- P_i had broadcasted $(r_{ij}^1, \ldots, r_{ij}^\ell)$ during **Round 2** of $\mathsf{Ver}(P_i, P_j, \mathcal{P}, (r_{ij}^1, \ldots, r_{ij}^\ell), \epsilon')$.
- D had broadcasted $(f_i^1(j), \ldots, f_i^\ell(j))$ during **Round 2** of $\mathsf{Ver}(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)), \epsilon')$.

Round 4:

1. For every **conflicting pair** (P_i, P_j), P_i reveals $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ and P_j reveals $ICSig(D, P_j, \mathcal{P}, (f_j^1(i), \ldots, f_j^\ell(i)))$, each having ϵ' error.
2. P_i reveals $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ having ϵ' error, for every $j = 0, \ldots, n$ if $(f_i^l(0), \ldots, f_i^l(n))$ do not define a degree t polynomial for some $l \in \{1, \ldots, \ell\}$.
3. Both the above steps will continue in **Round 5** as well.

Local Computation At the end of Round 5 (By Every Party)

1. D will be **discarded**, if one of the following happens:
   - For a **conflicting pair** (P_i, P_j), both P_i and P_j are successful in revealing $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ and $ICSig(D, P_j, \mathcal{P}, (f_j^1(i), \ldots, f_j^\ell(i)))$ respectively AND $f_i^l(j) \neq f_j^l(i)$ for some $l \in \{1, \ldots, \ell\}$.
   - P_i is successful in revealing $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ for every $j = 0, \ldots, n$ AND $(f_i^l(0), \ldots, f_i^l(n))$ revealed by P_i, do not define a degree t polynomial for some $l \in \{1, \ldots, \ell\}$.
2. If D is not discarded, then every P_i computes $ICSig(P_j, P_i, \mathcal{P}, (b_{ji}^1 - r_{ji}^1, \ldots, b_{ji}^\ell - r_{ji}^\ell)$ (which is same as $ICSig(P_j, P_i, \mathcal{P}, (f_j^1(i), \ldots, f_j^\ell(i))))$ for every P_j such that (P_i, P_j) is **not** a **conflicting pair**. Every party computes verification information corresponding to $ICSig(P_j, P_i, \mathcal{P}, (f_j^1(i), \ldots, f_j^\ell(i)))$. (Follows from Linearity of IC Signature presented in Section 2.5 of Chapter 2)

2. *The i^{th} values on the polynomials, namely $(s_i^1, \ldots, s_i^\ell)$, where $s_i^l = f^l(i)$ are either publicly known or otherwise party $P_i \in \mathcal{P}$ holds $ICSig(P, P_i, \mathcal{P}, (s_i^1, \ldots, s_i^\ell))$;*

Figure 4.4: Reconstruction Phase of $(2t+1, t)$ statistical VSS Scheme 5VSS-MS

5VSS-MS-Rec($D, \mathcal{P}, S, \epsilon$) — Two Rounds

Round 1 and 2: If D is not discarded in 5VSS-MS-Share, then every P_i reveals $ICSig(P_j, P_i, \mathcal{P}, (f_j^1(i), \ldots, f_j^\ell(i)))$ having ϵ' error, such that (P_i, P_j) is **not** a **conflicting pair**.

Local Computation at the end of Round 2 of 5VSS-MS-Rec (By Every Party)

1. Create a set REC. Add P_i to REC if:
   - P_i is successful in revealing $ICSig(P_j, P_i, \mathcal{P}, (f_j^1(i), \ldots, f_j^\ell(i)))$ for all P_j such that (P_i, P_j) is **not** a **conflicting pair AND**
   - For $l = 1, \ldots, \ell$, $(f_{i1}^l, \ldots, f_{in}^l)$ define degree t polynomial, say $f_i^l(x)$ where:
     - f_{ij}^l is equal to $f_j^l(i)$ revealed by P_i in reconstruction phase when (P_i, P_j) is **not** a **conflicting pair**;
     - $f_{ij}^l = f_i^l(j)$ when (P_i, P_j) is a **conflicting pair** and P_i had successfully revealed $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ during 5VSS-MS-Share.
2. For every $l = 1, \ldots, \ell$, reconstruct a symmetric bivariate polynomial of degree t in x and y, say $F^l(x, y)$, such that $F^l(x, i) = f_i^l(x)$ for all $P_i \in REC$.
3. Output $s^l = F^l(0, 0)$ for all $l = 1, \ldots, \ell$.

3. *For at most t i's $(s_i^1, \ldots, s_i^\ell)$ are publicly known.*

The $1d^{(\star,\ell)}$-sharing of S is denoted by $\langle S \rangle_t$. If some specific party P does the sharing, then we denote it by $\langle S \rangle_t^P$.

Notice that at the end of 5VSS-MS-Share, every honest P_i has done $1d^{(\star,\ell)}$-sharing of values $(f_i^1(0), \ldots, f_i^\ell(0))$ using polynomials $f_i^1(x), \ldots, f_i^\ell(x)$. Apart from this, D also has done $1d^{(\star,\ell)}$-sharing of values $(f_i^1(0), \ldots, f_i^\ell(0))$ for all $i = 1, \ldots, n$ using the same polynomials.

Definition 4.22 **($2d^{(\star,\ell)}$-sharing)** *A set of values $S = (s^1, \ldots, s^\ell) \in \mathbb{F}^\ell$ is $2d^{(\star,\ell)}$-shared among the parties in $\mathcal{P}$, denoted as $\langle\langle S \rangle\rangle_t$, if the following holds:*

1. *There exists degree t polynomials $f^l(x), f_1^l(x), \ldots, f_n^l(x)$ with $f^l(0) = s^l$, for $l = 1, \ldots, \ell$ and for $i = 1, \ldots, n$, $f_i^l(0) = f^l(i)$;*
2. *Every honest party $P_i \in \mathcal{P}$ holds a share $s_i^l = f^l(i)$ of s^l and the polynomial $f_i^l(x)$ for $l = 1, \ldots, \ell$. Moreover P_i has $1d^{(\star,\ell)}$-shared the values $(s_i^1, \ldots, s_i^\ell)$ using polynomials $(f_i^l(x), \ldots, f_i^\ell(x))$, .*

If some specific party P does the sharing, then we denote it by $\langle\langle S \rangle\rangle_t^P$.

As before, we can easily see that at the end of 5VSS-MS-Share, the secret $S = (s^1, \ldots, s^\ell)$ with $s^l = F^l(0,0)$ is $2d^{(\star,\ell)}$-shared by D where $f^l(x) = F^l(x,0)$ and $f_i^l(x) = F^l(x,i)$. The secret S can be reconstructed robustly from its $2d^{(\star,\ell)}$-sharing using the reconstruction phase protocol 5VSS-MS-Rec. Again as in the case of $2d^\star$-sharing, in the definition of $2d^{(\star,\ell)}$-sharing, we do not capture D's $1d^{(\star,\ell)}$-sharing of $(f_i^1(0), \ldots, f_i^\ell(0))$ for every $i = 1, \ldots, n$. These sharing are not required in 5VSS-MS-Rec for the reconstruction of the secrets. But we will use the $1d^{(\star,\ell)}$-sharing done by D in our multiplication protocol in order to either reconstruct $(f_i^1(0), \ldots, f_i^\ell(0))$ or detect that D is faulty. This again follows from the fact that a secret can not be reconstructed robustly from its $1d^{(\star,\ell)}$-sharing. But if the reconstruction fails then it can be concluded that D who has done the $1d^{(\star,\ell)}$-sharing is corrupted.

An inherent disadvantage of $2d^{(\star,\ell)}$-sharing of some secret S is that reconstruction of individual elements of S is not permitted. That is, all the elements of S will be reconstructed simultaneously even though a subset of the values in S is desired to be reconstructed. As and when we require, we will show how to deal with this problem.

There are $\Theta(n)$ and $\Theta(n^2)$ underlying IC signatures in $1d^{(\star,\ell)}$-sharing and $2d^{(\star,\ell)}$-sharing respectively. We now give the following definitions:

Definition 4.23 ($1d^{(\star,\ell)}$-sharing with ϵ Error) *We say that a $1d^{(\star,\ell)}$-sharing has ϵ error, if each of its $\Theta(n)$ underlying IC signature has $\frac{\epsilon}{n}$ error.*

We will show in Section 5.3.1 (of next chapter) that given $1d^{(\star,\ell)}$-sharing of ℓ secrets having ϵ error, the ℓ secrets can be reconstructed from its $1d^{(\star,\ell)}$-sharing, except with error probability ϵ when the party P who has done the sharing is honest. Additionally, if the secrets are not reconstructed, then except with error probability ϵ, party P who has done the sharing is corrupted and all honest parties will come to know this publicly.

Definition 4.24 ($2d^{(\star,\ell)}$-sharing with ϵ Error) *We say that a $2d^{(\star,\ell)}$-sharing has ϵ error, if each of its $\Theta(n^2)$ underlying IC signature has $\frac{\epsilon}{n^2}$ error.*

If a $2d^{(\star,\ell)}$-sharing is generated from 5VSS-MS-Share executed with error probability ϵ, then the resultant $2d^{(\star,\ell)}$-sharing will have ϵ error. From the proof of **Correctness** and **Strong Commitment** properties of 5VSS-MS, given $2d^{(\star,\ell)}$-sharing of ℓ secrets having ϵ error, the ℓ secrets can be reconstructed from its $2d^{(\star,\ell)}$-sharing, except with error probability ϵ. Hence we have the following important theorem.

Theorem 4.25 *Let $\langle\langle S \rangle\rangle_t$ be a $2d^{(\star,\ell)}$-sharing, having ϵ error. Then S can be correctly reconstructed from $\langle\langle S \rangle\rangle_t$, except with error probability ϵ.*

In the next section, we discuss the linearity property of $1d^{(\star,\ell)}$-sharing and $2d^{(\star,\ell)}$-sharing.

4.3.2 Linearity Property of $1d^{(\star,\ell)}$-sharing and $2d^{(\star,\ell)}$-sharing

Linearity Property of $1d^{(\star,\ell)}$-sharing: Now similar to the linearity of $1d^\star$-sharing, $1d^{(\star,\ell)}$-sharing also satisfies linearity property. Specifically, let $P \in \mathcal{P}$ be a party, who has done $1d^{(\star,\ell)}$-sharing of q sets of ℓ secrets, say $S^1, \ldots, S^q$, where

$S^j = (s^{1j}, \ldots, s^{\ell j})$. Moreover, let the following condition holds, which we call as *condition for linearity of* $1d^{(\star,\ell)}$*-sharing*: *For* $i = 1, \ldots, n$, *the IC signatures* $ICSig(P, P_i, \mathcal{P}, S_i^1), \ldots, ICSig(P, P_i, \mathcal{P}, S_i^q)$ *(possibly some of* S_i^j*'s are public) satisfy the condition of linearity of IC signatures (recall from Section 2.5 of Chapter 2), where* $S_i^1, \ldots, S_i^q$ *denotes the* i^{th} *share of* $S^1, \ldots, S^q$ *respectively.* Then the parties can compute $1d^{(\star,\ell)}$-sharing of $S = S^1 + \ldots + S^q$ without doing any further communication, where $S = (s^1, \ldots, s^q)$ and $s^l = \sum_{j=1}^{q} s^{lj}$. This is achieved by asking each party P_i to compute $ICSig(P, P_i, \mathcal{P}, S_i^1 + \ldots + S_i^q)$ from $ICSig(P, P_i, \mathcal{P}, S_i^1), \ldots, ICSig(P, P_i, \mathcal{P}, S_i^q)$ (this is possible as the signatures satisfy the *condition of linearity of IC signatures*), where $S_i^1 + \ldots + S_i^q$ denotes the i^{th} share of S. We capture this scenario by writing $\langle S \rangle_t^P = \sum_{i=1}^{q} \langle S_i \rangle_t^P$.

Again as in the case of $1d^\star$-sharing, linearity property does not apply on $1d^\star$-sharing when the sharings are generated by different parties. That is, given $\langle S^1 \rangle_t^P$ and $\langle S^2 \rangle_t^Q$, where P and Q are two different parties, then the parties cannot locally compute $1d^\star$-sharing of $S^1 + S^2$. This is because the underlying IC signatures in $\langle S^1 \rangle_t^P$ and $\langle S^2 \rangle_t^Q$ will not satisfy the condition of linearity of IC signatures, as stated in Note 2.14 at the end of Section 2.5 of Chapter 2.

Linearity Property of $2d^{(\star,\ell)}$*-sharing:* Now similar to the linearity property of $2d^\star$-sharing, $2d^{(\star,\ell)}$-sharing satisfies linearity property. Specifically, let $P \in \mathcal{P}$ be a party, who has done $2d^{(\star,\ell)}$-sharing of a number of secrets, say $S^1, \ldots, S^q$ each containing ℓ values. Moreover, let the following condition holds which we call as *condition for linearity of* $2d^{(\star,\ell)}$*-sharing*: *For every honest* P_i, *the* $1d^{(\star,\ell)}$*-sharing* $\langle S_i^1 \rangle_t^{P_i}, \ldots, \langle S_i^q \rangle_t^{P_i}$ *satisfies the condition for linearity of* $1d^{(\star,\ell)}$*-sharing. Here* $S_i^1, \ldots, S_i^q$ *denotes the* i^{th} *share of* $S^1, \ldots, S^q$ *respectively.* Then the parties can compute $2d^{(\star,\ell)}$-sharing of $S = S^1 + \ldots + S^q$ without doing any further communication. This is achieved by asking the parties to locally compute $\langle S_i^1 + \ldots + S_i^q \rangle_t^{P_i}$ from $\langle S_i^1 \rangle_t^{P_i}, \ldots, \langle S_i^q \rangle_t^{P_i}$ for all i, where $S_i^1 + \ldots + S_i^q$ denotes the i^{th} share of S. We capture this scenario by writing $\langle\langle S \rangle\rangle_t^P = \sum_{i=1}^{q} \langle\langle S^i \rangle\rangle_t^P$.

Unlike $1d^{(\star,\ell)}$-sharing, we can apply linearity property on $2d^{(\star,\ell)}$-sharing even when the sharings are generated by different parties, provided the underlying $1d^{(\star,\ell)}$-sharing satisfies the *condition for linearity of* $1d^{(\star,\ell)}$*-sharing.* More specifically, let S^1 and S^2 be secrets (each containing ℓ values) which are $2d^{(\star,\ell)}$-shared by two different parties, say P and Q; i.e., $\langle\langle S^1 \rangle\rangle_t^P$ and $\langle\langle S^2 \rangle\rangle_t^P$ are given. Moreover, for every honest P_i, let the underlying $1d^{(\star,\ell)}$-sharing $\langle S_i^1 \rangle_t^{P_i}$ and $\langle S_i^2 \rangle_t^{P_i}$ satisfies the condition for linearity of $1d^{(\star,\ell)}$-sharing. Then the parties can compute $2d^{(\star,\ell)}$-sharing of $S = S^1 + S^2$ without doing any further communication. That is $\langle\langle S \rangle\rangle_t = \langle\langle S^1 \rangle\rangle_t^P + \langle\langle S^2 \rangle\rangle_t^Q$ is doable.

Finally before ending this section, we prove the following lemmas.

Lemma 4.26 *Assume that* $\langle S^1 \rangle_t^P, \ldots, \langle S^q \rangle_t^P$ *are* q *different* $1d^{(\star,\ell)}$*-sharing, each having* ϵ *error. Let* $\langle S \rangle_t^P = \sum_{j=1}^{q} \langle S^j \rangle_t^P$. *Then* $\langle S \rangle_t^P$ *will have* ϵ *error.*

PROOF: The proof is similar to the proof of Lemma 4.16. □

Lemma 4.27 *Assume that* $\langle\langle S^1 \rangle\rangle_t, \ldots, \langle\langle S^q \rangle\rangle_t$ *are* q *different* $2d^{(\star,\ell)}$*-sharing, each having* ϵ *error. Let* $\langle\langle S \rangle\rangle_t = \sum_{j=1}^{q} \langle\langle S^j \rangle\rangle_t$. *Then* $\langle\langle S \rangle\rangle_t$ *will have* ϵ *error.*

The above lemma implies that the individual S^j's as well as the sum value S can be reconstructed from their corresponding $2d^{(\star,\ell)}$-sharing, except with error probability ϵ.

In the next chapter, when we use linearity property of $1d^{(\star,\ell)}$-sharing on a number of $1d^{(\star,\ell)}$-sharings, we assume that the *condition for linearity of* $1d^{(\star,\ell)}$-*sharing* has been satisfied. The same applies in case of $2d^{(\star,\ell)}$-sharing as well.

4.4 Conclusion and Open Problems

In this chapter, we designed statistical VSS with optimal resilience i.e with $n = 2t+1$ parties that achieves the best known communication and round complexity in the literature. So a natural open question is:

Open Problem 7 *Can we improve the round and communication complexity of statistical VSS with optimal resilience over the complexities that we provided in this chapter?*

Chapter 5

Statistical MPC with Optimal Resilience Minimizing both Round and Communication Complexity

In this chapter, we focus on statistical MPC with optimal resilience (i.e $n = 2t+1$ parties) in synchronous network (assuming the availability of a broadcast channel, in addition to point to point secure channel between every two parties).

The round and communication complexity are the most important complexity measures of MPC protocols in synchronous networks. A proper balance of both the complexity measures is essential from the perspective of practical implementation of MPC protocol. So far communication complexity wise the best known optimally resilient statistical MPC is reported in [12]. The protocol of [12] achieves $\mathcal{O}(n^2 \log \frac{1}{\epsilon})$ bits of private communication [5] per multiplication gate at the cost of high round complexity of $\mathcal{O}(n^2\mathcal{D})$, where $\mathcal{D}$ is the multiplicative depth of the arithmetic circuit representing function f. On the other hand, round complexity wise best known optimally resilient statistical MPC protocols are presented in [4, 5] and [138][6]. The protocols of [4, 5] and [138] have round complexity of $\mathcal{O}(\mathcal{D})$. But unfortunately, these MPC protocols require broadcasting[7] of $\Omega(n^5(\log \frac{1}{\epsilon})^4)$ bits per multiplication gate[8].

In this work, we focus to balance both the complexity measures of statistical MPC. With this aim in mind, we present a new optimally resilient statistical MPC that acquires a round complexity of $\mathcal{O}(\mathcal{D})$ and broadcasts $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits per multiplication gate. Hence our protocol maintains the round complexity of most round efficient protocol while improving the communication complexity. Moreover, for all functions with constant multiplicative depth, our protocol achieves constant round complexity while most communication efficient MPC of [12] requires $\mathcal{O}(n^2)$ rounds.

The key tools of our new MPC are the ICP presented in Chapter 2, the

[5]Communication over secure channels.

[6]We have considered MPC protocols with polynomial (in n and $\log \frac{1}{\epsilon}$) communication complexity. Constant round MPC can be achieved following the approach of [3] but at the expense of exponential blow-up in communication complexity.

[7]Communication over broadcast channel

[8]The authors of [49] claimed to have an optimally resilient statistical MPC protocol with round complexity of $\mathcal{O}(\mathcal{D})$ and communication complexity of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits of broadcast per multiplication gate, without providing exact implementation details.

statistical VSS protocol presented in Chapter 4 and a new, *robust* multiplication protocol for generating multiplication triples (that uses our VSS protocol of Chapter 4 as building block).

5.1 Introduction

5.1.1 Definition of MPC

MPC [151] allows a set of n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ to securely compute an agreed function f, even if some of the parties are under the control of a centralized active adversary. More specifically, assume that f can be expressed as $f : \mathbb{F}^n \rightarrow \mathbb{F}^n$ and party P_i has input $x_i \in \mathbb{F}$, where $\mathbb{F}$ is a finite field. Now MPC ensures the following:

1. **Correctness:** At the end of the computation of f, each honest P_i gets $y_i \in \mathbb{F}$, where $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, irrespective of the behavior of the corrupted parties.
2. **Secrecy:** Moreover, the adversary should not get any information about the input and output of the honest parties, other than what can be inferred from the input and output of the corrupted parties.

In any general MPC protocol, the function f is specified by an arithmetic circuit over $\mathbb{F}$, consisting of input, linear (e.g. addition), multiplication, random and output gates. We denote the number of gates of each type by c_I, c_A, c_M, c_R and c_O, respectively. *Among all the different types of gate, the evaluation/computation of a multiplication gate requires the most communication complexity. So the communication complexity of any general MPC is usually given in terms of the communication complexity per multiplication gate* [14, 13, 12, 52, 126].

5.1.2 The Relevant Literature on MPC

MPC protocol tolerating an active (i.e Byzantine) adversary controlling at most t out of n parties is possible if and only if $t < \frac{n}{3}$ [20]. This bound on resilience is optimal for error free computation. MPC without any error in computation is called as *perfectly secure* MPC (or in short *perfect* MPC). If a negligible error probability is allowed in the computation and a common broadcast channel is available then the resilience can be improved to $t < \frac{n}{2}$ [138, 4, 6]. MPC with negligible error probability in computation is called as *statistically secure* MPC (or in short *statistical* MPC). Moreover, statistical MPC designed with exactly $n = 2t + 1$ parties (in the presence of a broadcast channel, along with point to point communication between every two parties) is said to have optimal resilience.

5.1.3 Statistical MPC with Optimal Resilience

Statistical MPC with optimal resilience was first reported in [138] and in [4] independently. Subsequently, statistical MPC protocols with optimal resilience are reported in [138, 4, 3, 6, 48, 49, 12]. The main tools for designing any statistical MPC with optimal resilience are:

1. ICP [138, 48], which provides a way to authenticate information in the presence of computationally unbounded powerful active adversary;

2. VSS [43, 138, 48, 49], which allows a party to share/commit some secret such that the secret can be later reconstructed robustly;

3. ABC protocol [6], where a party proves $C = A.B$ after committing A, B and C;

4. Multiplication protocol, where the parties generate sharing of c from the sharing of a and b satisfying $c = ab$ and

5. Fault handling mechanism in multiplication protocol which is to be executed when some corrupted party(ies) is (are) detected to misbehave during multiplication protocol.

The optimally resilient statistical MPC protocols reported so far [138, 4, 6, 48, 49, 12] differ from each other in different implementations of the above tools. In the following, we briefly discuss about each of the above tools along with the citations of corresponding works.

Before doing that, we would like to clarify that in our discussion we have considered statistical MPC protocols with optimal resilience, having polynomial (in n and $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$, where ϵ is the error probability) communication complexity and polynomial (in n and $\mathcal{D}$, where $\mathcal{D}$ is the multiplicative depth of the circuit) round complexity. Applying the methods of [3], the number of rounds of MPC can be reduced to constant but at the expense of exponential blow-up in communication complexity. Hence, we will not consider the work of [3] in our discussion.

ICP is already discussed in detail in Chapter 2. We just recall that ICP was first introduced by Rabin et al. [138, 137] in order to design a statistical VSS protocol. Subsequently, a more efficient ICP was reported in [48].

VSS is a fundamental primitive used in many secure distributed computing protocols including MPC. Statistical VSS assuming $n = 2t + 1$ parties and a common broadcast channel is already discussed in Chapter 4. We just recall the following: Statistical VSS assuming $n = 2t + 1$ parties and a common broadcast channel was first reported in [138, 137]. Later more efficient statistical VSS protocols with $n = 2t + 1$ are proposed in [48] and [49].

In ABC protocol, a party who is committed, in some manner, to values a, b, and c, demonstrates in zero-knowledge that a, b, and c satisfy $c = ab$ without revealing their values and any other extra information. ABC protocol is an important tool used in multiplication protocol of MPC. It uses VSS as a black box. There are three different ABC protocols presented in [138], [4, 6] and [48] among which the protocol of [48] is the most efficient and simple. While ABC protocol of [48] requires $\mathcal{O}(1)$ invocations of VSS, both the protocols of [138] and [4, 6] require $\mathcal{O}(\kappa)$ invocations of VSS where $\kappa = \log \frac{1}{\epsilon}$ and ϵ is the error probability of the ABC and MPC protocol.

In multiplication protocol, given sharing of a and b, parties have to compute sharing of c where $c = ab$. Multiplication protocol is a major component of MPC protocol. It is generally accomplished by n invocations to ABC protocol along with some additional techniques. Though ABC protocols of [138] and [4, 6] are different, [138] and [4, 6] adapt almost the same techniques of polynomial randomization and degree reduction proposed by [20] to generate random sharing of c. Later [93] proposed an elegant method of generating random sharing of c using Vandermonde matrix. This idea later evolved to the more simpler idea of using Langrange's Interpolation formula (see [46] for more details).

Fault handling mechanism in multiplication protocol deals with the way the multiplication protocol reacts when some corrupted party is caught being misbehaving. It is generally achieved in three ways. If a party is detected to be faulty, then

1. In the first approach, the party is eliminated from computation from the time it was detected as corrupt and all the shared values are re-shared using lesser degree polynomial and after this the computation goes on [4]. Here the overhead is the cost of re-sharing each of the value that were shared at the time of fault detection;

2. In the second approach, the values possessed by the corrupted party are reconstructed and then all the parties publicly simulate the task of the corrupted party for the remaining execution of the protocol [138]. The overhead here is the cost of reconstructing the values of the corrupted party;

3. In yet another approach, the party is eliminated from computation and the computation restarts from the beginning. Here the overhead is the repetitive executions of same protocol [48].

Now depending on the implementation of the above techniques along with implementation of required sub-protocols, we may decide which of the three techniques will lead to better communication and round complexity.

So far in the literature, there are mainly two paradigms for designing MPC protocol:

1. *Input-Computation-Output Paradigm:* This paradigm is alternatively known as Share-Compute-Reveal paradigm [5]. As per this paradigm, an MPC protocol is structured into three phases, namely Input, Computation and Output Phase. The parties secretly share their inputs in Input Phase, run sub-protocols to evaluate gates (mainly addition and multiplication) of a bounded fan-in arithmetic circuit that expresses the function f in Computation phase, and reveal the final secret representing the output to appropriate parties in Output phase. The MPC protocols of [20, 41, 138, 4, 48] follow this paradigm.

2. *Preparation-Input-Computation-Output Paradigm:* This outstanding round-reducing paradigm was proposed by Beaver [5]. In brief, this paradigm simplifies the evaluation of multiplication gate during computation phase by performing some tasks well in advance in Preparation Phase (also called as preprocessing phase). This new technique of evaluating multiplication gate is more popularly known as Beaver's Circuit Randomization Technique [5]). According to this paradigm, sharing of c_M multiplication triples (x_i, y_i, z_i) (x_i and y_i are random and $z_i = x_i y_i$) is generated in preparation phase, every multiplication gate is associated with one multiplication triple and later during computation phase each multiplication gate is evaluated using its associated multiplication triple at the cost of reconstructions of two sharing. The protocol used for evaluating multiplication gate in Input-Computation-Output Paradigm can be used to generate sharing of z_i from the sharing of x_i and y_i. But the advantage of this paradigm over the previous one is that in this paradigm we can generate all the c_M multiplication triples in parallel where as in previous paradigm the multiplication gates have to be

evaluated sequentially as per the dependency relation of the circuit. As a result, if M is the number of rounds required to perform multiplication, then MPC designed following Input-Computation-Output paradigm will require $\mathcal{O}(M\mathcal{D})$ rounds, where as MPC following Preparation-Input-Computation-Output paradigm will require only $\mathcal{O}(M+\mathcal{D})$ rounds, as the reconstruction of shared values requires only constant number of rounds.

After the invention of Preparation-Input-Computation-Output paradigm, it has been used frequently in many MPC protocols that appeared subsequently [98, 101, 49, 12, 52, 14].

In comparison to the other optimally resilient statistical MPC protocols, the MPC protocol of [12] has received slightly different treatment. So we emphasize on it with little bit more detail. In line with the existing statistical MPC protocols, the authors of [12] have designed ICP, VSS, ABC and multiplication protocols and used Preparation-Input-Computation-Output paradigm in their MPC protocol. But contrary to the existing MPC protocols, all the above primitives are designed in *dispute control framework* (a generalization of *player elimination framework* introduced by [98]) where the implementations of the primitives are non-robust as they fail when at least one corrupted party misbehaves. In case of failure, the protocols output a dispute which is a pair of parties with at least one of them is guaranteed to be corrupted. During the course of the protocol, the parties keep track of disputes that arise among them, and the ongoing computation is adjusted such that known disputes cannot arise again. To keep the communication complexity low, the computation of each of the phases has been divided into n^2 segments. Each segment is executed in a non-robust manner, where a segment fails when one of the protocols invoked in it has failed with a dispute as output. If the computation of a segment passes then the next segment is taken up for computation; otherwise the same segment is recomputed again with the mechanism to prevent existing dispute to happen again. As there can be at most $nt = \mathcal{O}(n^2)$ possible pair of parties with at least one party in a pair being corrupted, there are $\mathcal{O}(n^2)$ possible disputes and thus the segments may fail $\mathcal{O}(n^2)$ times in total.

Now Table 5.1 summarizes the communication complexity and round complexity of known statistical MPC protocols with optimal resilience. In all these protocols, the computation is assumed to be done over a finite field $\mathbb{F} = GF(2^\kappa)$, where $\epsilon = 2^{-\Omega(\kappa)}$ and ϵ is the error probability. Thus each field element is represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

5.1.4 Our Motivation and Contribution

Round complexity and communication complexity are the two important complexity measures of any fault-tolerant distributed computing protocol such as MPC. In distributed systems, communication is usually expensive, and protocols designed for practical use must use as few rounds of communication, with as small messages as possible. Analyzing Table 5.1, we find that researchers have reduced the communication complexity of MPC protocol considerably but at the expense of high round complexity [48, 49, 12]. This trend is undesirable if we ever hope to implement MPC protocols in practice. So motivated to design MPC protocol that minimizes both round and communication complexity simultaneously, we present a new statistical MPC protocol with optimal resilience, that achieves a

Table 5.1: Communication Complexity and Round Complexity of Existing statistical MPC protocols with Optimal Resilience.

Reference	Communication Complexity in bits	Round Complexity
[138]	Private: $\Omega(c_M n^5 (\log \frac{1}{\epsilon})^4)$; Broadcast: $\Omega(c_M n^5 (\log \frac{1}{\epsilon})^4)$	$\mathcal{O}(\mathcal{D})$
[4, 6]	Private: $\Omega(c_M n^5 (\log \frac{1}{\epsilon})^4)$; Broadcast: $\Omega(c_M n^5 (\log \frac{1}{\epsilon})^4)$	$\mathcal{O}(\mathcal{D})$
[48]	Private: $\mathcal{O}(c_M n^5 \log \frac{1}{\epsilon})$; Broadcast: $\mathcal{O}(c_M n^5 \log \frac{1}{\epsilon})$	$\mathcal{O}(n\mathcal{D})$
[49][a]	Private: $\mathcal{O}(c_M n^5 \log \frac{1}{\epsilon})$; Broadcast: $\mathcal{O}(c_M n^5 \log \frac{1}{\epsilon})$	$\mathcal{O}(n + \mathcal{D})$
[12]	Private: $\mathcal{O}(c_M n^2 \log \frac{1}{\epsilon})$; Broadcast: $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$	$\mathcal{O}(n^2\mathcal{D})$

[a] The authors have actually claimed to have a protocol with a communication complexity of $\mathcal{O}(c_M n^4 \kappa)$ bits (private communication plus broadcast) and round complexity of $\mathcal{O}(\mathcal{D})$ without providing any details.

communication complexity of $\mathcal{O}(c_M n^3 \log \frac{1}{\epsilon})$ bits for both private and broadcast communication while maintaining a round complexity of $\mathcal{O}(\mathcal{D})$. At the heart of our new statistical MPC protocol are:

1. The ICP presented in Chapter 2, that provides the best known round and communication complexity in the literature.

2. The statistical VSS presented in Chapter 4, that provides the best known round and communication complexity in the literature.

3. An efficient and novel multiplication protocol with robust fault handling mechanism (that uses our statistical VSS of Chapter 4 as building block).

Our statistical MPC protocol involves a negligible error probability of ϵ in correctness property. To bound the error probability by ϵ, all computation in our protocol are performed over a finite field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^3 2^{-\kappa} \cdot \max(n^2, (2c_M + c_O))$. We assume that $n = \text{poly}(c_M, c_O, c_A, c_I)$. Any field element from field $\mathbb{F}$ can be represented by κ bits, where $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ (this can be derived using $n = \mathcal{O}(\log \frac{1}{\epsilon})$ and $n = \text{poly}(c_M, c_O, c_A, c_I)$).

In order to bound the error probability of our MPC protocol by some specific value of ϵ, we find out the *minimum* value of κ that satisfies $\epsilon \geq n^3 2^{-\kappa} \cdot \max(n^2, (2c_M + c_O))$. This value for κ will consequently determine the field $\mathbb{F}$ over which our protocol should work.

5.1.5 Our Network and Adversary Model

The network and adversary model is same as the one presented in in Section 2.1.2 of Chapter 2. Recall that the set of parties is denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$ and t out of the n parties can be under the influence of a *computationally unbounded powerful, Byzantine (active), rushing adversary*, denoted as $\mathcal{A}_t$. Apart from pairwise secure channels, there is a physical broadcast channel available in the network. In this chapter, we assume $n = 2t + 1$.

5.1.6 The Road-map

Section 5.2 briefly discusses the overview of our statistical MPC protocol. Subsequently, Section 5.3, Section 5.4, Section 5.5 and Section 5.6 present the protocols for our proposed MPC protocol. Specifically, Section 5.3 contains our robust multiplication protocol. Lastly, we conclude this chapter in Section 5.7.

5.2 Overview of Our Statistical MPC Protocol

Our statistical MPC protocol is a sequence of following three phases: preparation phase, input phase and computation phase. In the preparation phase, $2d^{\star}$-sharing (recall from Chapter 4) of $c_M + c_R$ random multiplication triples will be generated. A triple (a, b, c) is called random multiplication triple if a and b are random and $c = ab$ holds. Our preparation phase requires only constant number of rounds. Each multiplication gate and random gate of the circuit will be associated with a $2d^{\star}$-sharing of random multiplication triple. In the input phase the parties $2d^{\star}$-share their inputs. In the computation phase, based on the inputs of the parties, the actual circuit will be computed gate by gate, such that the outputs of the intermediate gates are always kept as secret and are properly $2d^{\star}$-shared among the parties. Due to the linearity of the used $2d^{\star}$-sharing, the parties can locally evaluate linear gates without doing any communication. Each multiplication gate will be evaluated with the help of the multiplication triple associated with it, using the so called Beaver's circuit randomization technique [5].

In the preparation phase efficient multiplication protocol with robust fault handling has been used to generate multiplication triples. Our multiplication protocol is new and uses several subtle ideas to achieve its goal.

5.3 Preparation Phase

In the preparation phase, we generate $2d^{\star}$-sharing of $c_M + c_R$ random multiplication triples $((a^k, b^k, c^k)\ ;\ k = 1, \ldots, c_M + c_R)$, where $c^k = a^k b^k$. Moreover, each of the sharing will have $\frac{\epsilon}{(2c_M + c_O)}$ error (the reason for the selection of error will be clear later during the discussion of Computation phase; Also recall Definition 4.14 for the interpretation of the statement). We will proceed step by step. First using our multiplication protocol, we will generate $2d^{(\star, c_M + c_R)}$-sharing of all the a's, namely $\langle\langle A \rangle\rangle_t$ where $A = (a^1, \ldots, a^{c_M + c_R})$ and likewise $\langle\langle B \rangle\rangle_t$ and $\langle\langle C \rangle\rangle_t$ where $B = (b^1, \ldots, b^{c_M + c_R})$ and $C = (c^1, \ldots, c^{c_M + c_R})$. After that we will generate separate $2d^{\star}$-sharing of each of the individual values from the above $2d^{(\star, c_M + c_R)}$-sharings and thereby meet our requirement. We would like to clarify that in our multiplication protocol we could have generated $2d^{\star}$-sharing of a^k and b^k directly using 5VSS-Share (that deals with single secret; presented in Chapter 4) and then compute $2d^{\star}$-sharing of c^k from $2d^{\star}$-sharing of a^k and b^k. But this will require more communication complexity than our roundabout approach of generating $2d^{\star}$-sharing of these values.

5.3.1 Multiplication Protocol With Robust Fault Handling

Before presenting our multiplication protocol, we describe a few tools/sub-protocols which will be used as building blocks for our multiplication protocol.

5.3.1.1 Generating Random $2d^{(\star,\ell)}$-Sharing

Here we present a protocol called Random which allows the parties in $\mathcal{P}$ to jointly generate $2d^{(\star,\ell)}$-sharing of ℓ random values, i.e $\langle\langle r^1,\ldots,r^\ell\rangle\rangle_t$, about which $\mathcal{A}_t$ will have no information. The sharing $\langle\langle r^1,\ldots,r^\ell\rangle\rangle_t$ will have $\frac{\epsilon}{n}$ error. The protocol generates the sharing, except with error probability ϵ (we distinguish between the error associated with the sharing and the error probability of the protocol).

To bound the error probability by ϵ, the computation of Random is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^4 2^{-\kappa}$. This is derived from the fact that Random invokes $\mathsf{5VSS\text{-}MS\text{-}Share}$ with $\frac{\epsilon}{n}$ error probability and as mentioned in Section 4.3 (of Chapter 4), $\epsilon \geq n^3 2^{-\kappa}$ should hold to bound the error probability of $\mathsf{5VSS\text{-}MS}$ by ϵ. The protocol is now given in Fig. 5.1.

Figure 5.1: Protocol for Generating $2d^{(\star,\ell)}$-sharing of ℓ random values.

$$\langle\langle r^1,\ldots,r^\ell\rangle\rangle_t = \mathsf{\mathbf{Random}}(\mathcal{P},\ell,\epsilon)$$

1. Each $P_i \in \mathcal{P}$ selects ℓ random elements $r^{1i},\ldots,r^{\ell i}$ from $\mathbb{F}$ and then invokes $\mathsf{5VSS\text{-}MS\text{-}Share}(P_i,\mathcal{P},(r^{1i},\ldots,r^{\ell i}),\frac{\epsilon}{n})$ to generate $\langle\langle r^{1i},\ldots,r^{\ell i}\rangle\rangle_t^{P_i}$ having $\frac{\epsilon}{n}$ error.
2. Let $Pass$ denote the set of parties P_i in $\mathcal{P}$ such that $\mathsf{5VSS\text{-}MS\text{-}Share}(P_i,\mathcal{P},(r^{1i},\ldots,r^{\ell i}),\frac{\epsilon}{n})$ is executed successfully without P_i being discarded.
3. If $|Pass| \geq t+1$, all the parties in $\mathcal{P}$ jointly compute $\langle\langle r^1,\ldots,r^\ell\rangle\rangle_t = \sum_{P_i\in Pass}\langle\langle r^{1i},\ldots,r^{\ell i}\rangle\rangle_t^{P_i}$, where $r^l = \sum_{P_i\in Pass} r^{li}$, for $l = 1,\ldots,\ell$.

Lemma 5.1 *Protocol* Random *satisfies the following properties:*

1. **Correctness**: *Except with probability ϵ,* Random *outputs correct $2d^{(\star,\ell)}$-sharing of ℓ random values, having $\frac{\epsilon}{n}$ error.*
2. **Secrecy**: *The ℓ values whose $2d^{(\star,\ell)}$-sharing is generated by the protocol will be completely random and unknown to $\mathcal{A}_t$.*

PROOF: **Correctness:** As each instance of $\mathsf{5VSS\text{-}MS\text{-}Share}$ is invoked with error parameter $\frac{\epsilon}{n}$, corresponding to each party P_i in $Pass$, $\langle\langle r^{1i},\ldots,r^{\ell i}\rangle\rangle_t^{P_i}$ will have $\frac{\epsilon}{n}$ error. Now as $\langle\langle r^1,\ldots,r^\ell\rangle\rangle_t = \sum_{P_i\in Pass}\langle\langle r^{1i},\ldots,r^{\ell i}\rangle\rangle_t^{P_i}$, it follows from Lemma 4.27 that $\langle\langle r^1,\ldots,r^\ell\rangle\rangle_t$ will have $\frac{\epsilon}{n}$ error. Moreover the ℓ values shared by every honest party are random and there exists at least one honest party in $Pass$. This implies that the values $r^1,\ldots,r^\ell$ are random.

Now we show that Random will generate its output except with error probability ϵ. An honest P_i will be able to produce $\langle\langle r^{1i},\ldots,r^{\ell i}\rangle\rangle_t^{P_i}$ (having $\frac{\epsilon}{n}$ error) except with error probability $\frac{\epsilon}{n}$. This is because with probability at most $\frac{\epsilon}{n}$, honest P_i might get discarded during $\mathsf{5VSS\text{-}MS\text{-}Share}$ (see Claim 4.1) in which case P_i will not be included in $Pass$. Since there are $t+1$ honest parties, none of them will figure in $Pass$ with probability $(t+1)\frac{\epsilon}{n} \approx \epsilon$. This will result in $|Pass| \leq t$ and hence output will not be computed, with probability at most ϵ.

Hence, except with probability ϵ, Random will generate its desired output.

Secrecy: From the **Secrecy** property of 5VSS-MS-Share, the values which are $2d^{(\star,\ell)}$-shared by an honest party are completely random and are unknown to $\mathcal{A}_t$. $Pass$ will definitely contain at least one honest party. Now since addition preserves randomness, $r^1, \ldots, r^\ell$ will be completely random and unknown to $\mathcal{A}_t$. This proves **Secrecy** property. □

Lemma 5.2 *Protocol Random has the following bounds:*

1. **Round Complexity:** *Five Rounds.*
2. **Communication Complexity:** *Private and broadcast communication of $\mathcal{O}((\ell n^3 + n^4)\log\frac{1}{\epsilon})$ bits.*

PROOF: There are n parallel invocations of 5VSS-MS-Share and each invocation requires five rounds (see Theorem 4.19) and communication of $\mathcal{O}((\ell n^2+n^3)\log\frac{1}{\epsilon})$ bits, both private as well as broadcast (see Theorem 4.20). Hence the lemma. □

5.3.1.2 Public Reconstruction of $1d^{(\star,\ell)}$-sharing of ℓ Values

Let $P \in \mathcal{P}$ be a party, who has done $1d^{(\star,\ell)}$-sharing of a set of ℓ values, say $s^1, \ldots, s^\ell$. That is $\langle s^1, \ldots, s^\ell \rangle_t^P$ is given. Moreover, let the $1d^{(\star,\ell)}$-sharing has ϵ error. Then we present a protocol called Rec that tries to publicly reconstruct $s^1, \ldots, s^\ell$ from $\langle s^1, \ldots, s^\ell \rangle_t^P$.

By the definition of $1d^{(\star,\ell)}$-sharing having ϵ error, either P_i holds IC signature of P on the i^{th} shares of the secrets, where the IC signature will have $\frac{\epsilon}{n}$ error or the i^{th} shares are publicly known. By the definition of $1d^{(\star,\ell)}$-sharing, at most t such shares may be publicly known. That is, for $i = 1, \ldots, n$, either party P_i holds $ICSig(P, P_i, \mathcal{P}, (s_i^1, \ldots, s_i^\ell))$, having $\frac{\epsilon}{n}$ error or $(s_i^1, \ldots, s_i^\ell)$ is publicly known with the restriction that at most t such shares are public. Essentially, in protocol Rec the IC signatures held by the parties will be revealed. Then we check whether the values that are correctly revealed (from IC signature) and the values that are public already lie on a degree-t polynomial which should ideally hold according to the definition of $1d^{(\star,\ell)}$-sharing.

This protocol works over a field that was used by the protocol that generates the given $1d^{(\star,\ell)}$-sharing. Protocol Rec has the following features:

1. If P is honest then except with probability at most ϵ, the protocol will succeed to publicly reconstruct $s^1, \ldots, s^\ell$;
2. If the protocol fails to reconstruct the secrets then with probability at least $(1-\epsilon)$, party P is corrupted and every honest party will come to know that P is corrupted.

The protocol is formally given in Fig. 5.2.

Lemma 5.3 *Given $\langle s^1, \ldots, s^\ell \rangle_t^P$ having ϵ error, protocol Rec reconstructs the secrets $(s^1, \ldots, s^\ell)$ except with probability ϵ, when P is honest. On the other hand, if the protocol fails to reconstruct the secrets then except with probability ϵ, party P is corrupted.*

Figure 5.2: Public Reconstruction of ℓ Values that are $1d^{(\star,\ell)}$-shared by some party P.

$\mathsf{Rec}(\mathcal{P}, \ell, \langle s^1, \ldots, s^\ell \rangle_t^P, \epsilon)$

1. Given $\langle s^1, \ldots, s^\ell \rangle_t^P$ having ϵ error, either party P_i holds $ICSig(P, P_i, \mathcal{P}, (s_i^1, \ldots, s_i^\ell))$ having $\frac{\epsilon}{n}$ error, where s_i^l denotes i^{th} share of s^l OR $(s_i^1, \ldots, s_i^\ell)$ is known publicly.
2. Each party P_i holding $ICSig(P, P_i, \mathcal{P}, (s_i^1, \ldots, s_i^\ell))$, reveals $ICSig(P, P_i, \mathcal{P}, (s_i^1, \ldots, s_i^\ell))$ having $\frac{\epsilon}{n}$ error.
3. Each party P_j either reconstruct s^l for $l = 1, \ldots, \ell$ or decide that P is corrupted as follows:
   (a) For $l = 1, \ldots, \ell$, consider s_i^l values corresponding to all P_i's, who are successful in revealing the IC signature in step 2 and the s_i^l values which are already public and check whether they define a unique degree t polynomial. If yes then the constant term of the degree t polynomial is taken as s^l. Otherwise, P is decided to be corrupted.

PROOF: Since $\langle s^1, \ldots, s^\ell \rangle_t^P$ has ϵ error, it implies that either party P_i holds $ICSig(P, P_i, \mathcal{P}, (s_i^1, \ldots, s_i^\ell))$, having $\frac{\epsilon}{n}$ error or $(s_i^1, \ldots, s_i^\ell)$ is already public. Also by the definition of $1d^{(\star,\ell)}$-sharing, at most for t i's, $(s_i^1, \ldots, s_i^\ell)$ are public. Now we have the following two cases:

1. *Party P is honest*: In this case, each P_i who has succeeded to reveal IC signature will reveal correct $ICSig(P, P_i, \mathcal{P}, (s_i^1, \ldots, s_i^\ell))$ irrespective of whether it is honest or corrupted. Therefore the s_i^l values will lie on degree t polynomial and thus the secrets will be reconstructed correctly.

   However the above event has ϵ error probability. This is because, a corrupted P_i may reveal $ICSig(P, P_i, \mathcal{P}, (\overline{s_i^1}, \ldots, \overline{s_i^\ell}))$ with probability $\frac{\epsilon}{n}$ (according to **ICP-Correctness3**). Consequently, any one of the corrupted parties (there are at most t corrupted parties) may reveal incorrect IC signature with probability at most $t\frac{\epsilon}{n} \approx \epsilon$ (in which case, s_i^l values will not lie on degree t polynomial). Therefore an honest P may be proved as corrupted, with probability ϵ. Stating in other way, when the parties decides P to be corrupted, then it is true, with probability $(1 - \epsilon)$.

2. *Party P is corrupted*: In this case, each honest P_i will be able to successfully reveal correct $ICSig(P, P_i, \mathcal{P}, (s_i^1, \ldots, s_i^\ell))$, except with error probability $\frac{\epsilon}{n}$ from **ICP-Correctness2**. As there are at least $t+1$ honest parties, except with probability $(t+1)\frac{\epsilon}{n} \approx \epsilon$, all honest parties will successfully reveal correct IC signatures and therefore the s_i^l values revealed by them will lie on degree-t polynomials. However, since P is corrupted, a corrupted P_i in collaboration with the corrupted P can reveal any $ICSig(P, P_i, \mathcal{P}, (\overline{s_i^1}, \ldots, \overline{s_i^\ell}))$. Now if the s_i^l values revealed by corrupted P_i's along with the s_i^l values revealed by honest P_i's lie on a degree-t polynomial, then s^l will be reconstructed. Otherwise, everybody will come to know that party P is corrupted. □

Lemma 5.4 *Protocol* Rec *has the following bounds:*

1. **Round Complexity:** *Two rounds.*
2. **Communication Complexity:** *Broadcast of* $\mathcal{O}((\ell n + n^2)\log\frac{1}{\epsilon})$ *bits.*

PROOF: Follows from the following facts: (a) In Rec, there are $\Theta(t)$ parallel executions of Reveal; (b) Reveal requires two rounds and $\mathcal{O}((\ell + n)\log\frac{1}{\epsilon})$ bits of broadcast. □

5.3.1.3 ABC Protocol – Proving $c = ab$

Consider the following problem: let $P \in \mathcal{P}$ has generated $\langle(a^1, \ldots, a^\ell)\rangle_t^P$ and $\langle(b^1, \ldots, b^\ell)\rangle_t^P$, each having $\frac{\epsilon}{n}$ error. Now P wants to generate $\langle\langle(c^1, \ldots, c^\ell)\rangle\rangle_t^P$, having ϵ error, where $c^l = a^l b^l$ for $l = 1, \ldots, \ell$. Moreover, during this process, an honest P does not want to leak any *additional* information about a^l, b^l and c^l. Furthermore, if P is corrupted, then he may intentionally fail to generate the above output in which case every body will know that party P is corrupted.

We propose a protocol ProveCeqAB to achieve the above task. The protocol generates the correct output, except with error probability ϵ. The idea of the protocol is inspired from [48] with the following modification: we make use of our protocol 5VSS-MS (instead of their statistical VSS protocol), which provides us with high efficiency, both in terms of communication and round complexity.

We explain the idea of the protocol with a single pair (a, b). With respect to a single pair, the problem becomes like this: P has already $1d^\star$-shared a and b using degree-t polynomials, say $f_a(x)$ and $f_b(x)$. Now he wants to generate $2d^\star$-sharing of c, where $c = ab$, without leaking any *additional* information about a, b and c. To achieve this goal, P first selects a random non-zero $\beta \in \mathbb{F}$ and generates $2d^\star$-sharing of c, β and $d = \beta b$. Let $f_c(x), f_\beta(x)$ and $f_d(x)$ are polynomials implicitly used for sharing c, β and d. All the parties in $\mathcal{P}$ then jointly generate a random value r. P then broadcasts the polynomial $F(x) = rf_a(x) + f_\beta(x)$. Every party locally checks whether the appropriate linear combination of his shares lies on the broadcasted polynomial $F(x)$. If it does not satisfy then the party reveals P's signature on the shares of a and β. If the signatures are valid and indeed the party's value does not lie on $F(x)$, then all the parties will conclude that P fails to prove $c = ab$ and the protocol terminates here.

Otherwise, P again broadcasts $G(x) = F(0)f_b(x) - f_d(x) - rf_c(x)$. As before every party locally checks whether the appropriate linear combination of his shares lies on the broadcasted polynomial $G(x)$. If it does not satisfy then the party reveals P's IC signature on the shares of b, d and c. If the signatures are valid and indeed the party's value does not lie on $G(x)$, then all parties will conclude that P fails to prove $c = ab$ and the protocol terminates here. Otherwise every party checks whether $G(0) \stackrel{?}{=} 0$. If so then everybody accepts the $2d^\star$-sharing of c as valid $2d^\star$-sharing of ab. It is easy to check that $G(0)$ will be zero when P behaves honestly.

If a corrupted P shares $c \neq ab$, then the probability that $G(0) = 0$ holds is negligible because of the random r. This can be argued as follows: $G(0) = F(0)b - d - rc = (ra + \beta)b - d - rc = rab - rc + \beta b - d = r(ab - c) + \beta b - d$. Now if P shares $c \neq ab$ and $d \neq \beta b$, then $q = r(ab - c) + \beta b - d$ will be non-zero, except for only one value of r. But since r is randomly generated, the probability that r is that value is $\frac{1}{|\mathbb{F}|}$ which is negligibly small. The secrecy follows from the fact that $F(0)$ is randomly distributed and $G(0) = 0$. Protocol ProveCeqAB extends the above idea for ℓ pairs (a^l, b^l).

ProveCeqAB works on a field $\mathbb{F}$ which was used for protocol Random i.e $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^4 2^{-\kappa}$. This comes from the following facts: Since ProveCeqAB invokes Random with ϵ error probability, $\epsilon \geq n^4 2^{-\kappa}$ should hold. Moreover, ProveCeqAB invokes 5VSS-MS-Share with ϵ error probability which enforces $\epsilon \geq n^3 2^{-\kappa}$. Therefore, $\epsilon \geq \max(n^4 2^{-\kappa}, n^3 2^{-\kappa}) = n^4 2^{-\kappa}$ should hold for ProveCeqAB. Now the protocol is formally given in Fig. 5.3.

Lemma 5.5 *Protocol ProveCeqAB satisfies the following properties:*

1. **Correctness:** *If P is honest, then except with probability ϵ, P will be able to generate $\langle\langle(c^1, \ldots, c^\ell)\rangle\rangle_t^P$. If P is corrupted and the protocol succeeds then except with probability ϵ, P has generated $\langle\langle(c^1, \ldots, c^\ell)\rangle\rangle_t^P$, where $c^l = a^l b^l$, for $l = 1, \ldots, \ell$. Moreover, irrespective of whether P is honest or corrupted, if $\langle\langle(c^1, \ldots, c^\ell)\rangle\rangle_t^P$ is generated in the protocol then it will have ϵ error.*
2. **Secrecy:** *If P is honest then a^l, b^l, c^l will be information theoretically secure for all $l = 1, \ldots, \ell$.*

PROOF: **Correctness**: Notice that if at all $\langle\langle(c^1, \ldots, c^\ell)\rangle\rangle_t^P$ is generated in the protocol, then it will have ϵ error, irrespective of whether P is honest or corrupted. This is because $\langle\langle(c^1, \ldots, c^\ell)\rangle\rangle_t^P$ is generated by executing 5VSS-MS-Share$(P, \mathcal{P}, \mathcal{C}, \epsilon)$.

Next we show that if P is honest, then P would be successfully able to generate $\langle\langle(c^1, \ldots, c^\ell)\rangle\rangle_t^P$, except with probability ϵ. If P is honest, then except with error probability ϵ, he will not be discarded in 5VSS-MS-Share$(P, \mathcal{P}, \mathcal{C}, \epsilon)$, 5VSS-MS-Share$(P, \mathcal{P}, \mathcal{B}, \epsilon)$ and 5VSS-MS-Share$(P, \mathcal{P}, \mathcal{D}, \epsilon)$. The parties will jointly generate r, except with probability ϵ. Now from the protocol steps, it is clear that an honest party P may fail to prove $c = ab$, if some corrupted party P_i can forge one of the following IC signatures: $ICSig(P, P_i, \mathcal{P}, (fa^1(i), \ldots, fa^\ell(i)))$, $ICSig(P, P_i, \mathcal{P}, (f\beta^1(i), \ldots, f\beta^\ell(i)))$, $ICSig(P, P_i, \mathcal{P}, (fb^1(i), \ldots, fb^\ell(i)))$, $ICSig(P, P_i, \mathcal{P}, (fc^1(i), \ldots, fc^\ell(i)))$ and $ICSig(P, P_i, \mathcal{P}, (fd^1(i), \ldots, fd^\ell(i)))$. Since $(a^1, \ldots, a^\ell)$ and $(b^1, \ldots, b^\ell)$ are $1d^{(\star,\ell)}$-shared and each of these sharings has $\frac{\epsilon}{n}$ error, it implies that each of $ICSig(P, P_i, \mathcal{P}, (fa^1(i), \ldots, fa^\ell(i)))$ and $ICSig(P, P_i, \mathcal{P}, (fb^1(i), \ldots, fb^\ell(i)))$ will have $\frac{\epsilon}{n^2}$ error. Therefore a corrupted P_i can forge $ICSig(P, P_i, \mathcal{P}, (fa^1(i), \ldots, fa^\ell(i)))$ or $ICSig(P, P_i, \mathcal{P}, (fb^1(i), \ldots, fb^\ell(i)))$ with probability at most $\frac{\epsilon}{n^2}$. On the other hand, since $(\beta^1, \ldots, \beta^\ell)$, $(c^1, \ldots, c^\ell)$ and $(d^1, \ldots, d^\ell)$ are $2d^{(\star,\ell)}$-shared and each one of these sharing has ϵ error, it implies that each of $ICSig(P, P_i, \mathcal{P}, (f\beta^1(i), \ldots, f\beta^\ell(i)))$, $ICSig(P, P_i, \mathcal{P}, (fc^1(i), \ldots, fc^\ell(i)))$ and $ICSig(P, P_i, \mathcal{P}, (fd^1(i), \ldots, fd^\ell(i)))$ will have $\frac{\epsilon}{n^2}$ error. Hence a corrupted P_i can forge $ICSig(P, P_i, \mathcal{P}, (f\beta^1(i), \ldots, f\beta^\ell(i)))$ or $ICSig(P, P_i, \mathcal{P}, (fc^1(i), \ldots, fc^\ell(i)))$ or $ICSig(P, P_i, \mathcal{P}, (fd^1(i), \ldots, fd^\ell(i)))$ with probability at most $\frac{\epsilon}{n^2}$. Now as there can be t corrupted parties, any one of them can do the above forgeries with probability at most $t\frac{\epsilon}{n^2} \approx \frac{\epsilon}{n}$. Hence overall, if P is honest, then P would be successfully able to generate $\langle\langle(c^1, \ldots, c^\ell)\rangle\rangle_t^P$, except with probability ϵ.

Finally, we show that if a corrupted P has generated $\langle\langle(c^1, \ldots, c^\ell)\rangle\rangle_t^P$, then except with probability ϵ, $c^l = a^l b^l$ for $l = 1, \ldots, \ell$. This follows from the fact that if a corrupted P has generated $\langle\langle(c^1, \ldots, c^\ell)\rangle\rangle_t^P$, then $q^l = 0$ for all $l = 1, \ldots, \ell$. Now notice that $q^l = p^l b^l - d^l - rc^l = (ra^l + \beta^l)b^l - b^l\beta^l - rc^l = r(a^l b^l - c^l) + \beta^l b^l - d^l$. Now

Figure 5.3: Protocol to Generate $2d^{(\star,\ell)}$-sharing of $(c^1,\ldots,c^\ell)$ where $c^l = a^l b^l$ for $l = 1,\ldots,\ell$.

$$\langle\langle c^1,\ldots,c^\ell\rangle\rangle_t^P = \textsf{\textbf{ProveCeqAB}}(P,\mathcal{P},\langle a^1,\ldots,a^\ell\rangle_t^P,\langle b^1,\ldots,b^\ell\rangle_t^P,\epsilon)$$

1. Party P does the following:

   (a) Select ℓ non-zero random elements $\beta^1,\ldots,\beta^\ell$. For $l = 1,\ldots,\ell$, let $c^l = a^l b^l$ and $d^l = b^l\beta^l$. Let $\mathcal{B} = (\beta^1,\ldots,\beta^\ell)$, $\mathcal{C} = (c^1,\ldots,c^\ell)$ and $\mathcal{D} = (d^1,\ldots,d^\ell)$.

   (b) Invoke 5VSS-MS-Share$(P,\mathcal{P},\mathcal{C},\epsilon)$, 5VSS-MS-Share$(P,\mathcal{P},\mathcal{B},\epsilon)$ and 5VSS-MS-Share$(P,\mathcal{P},\mathcal{D},\epsilon)$. For $l = 1,\ldots,\ell$, let a^l, b^l, c^l, β^l and d^l are implicitly shared using degree-t polynomials $fa^l(x)$, $fb^l(x)$, $fc^l(x)$, $f\beta^l(x)$ and $fd^l(x)$ respectively.

2. If P is discarded during any of the three 5VSS-MS-Share protocols, then every party concludes that P fails to prove $c = ab$ and protocol terminates here.

3. Now all the parties in $\mathcal{P}$ jointly generate a random number r. This is done as follows: first the parties in $\mathcal{P}$ execute the protocol Random$(\mathcal{P},1,\epsilon)$ to generate $\langle\langle r\rangle\rangle_t$. Then the parties execute 5VSS-Rec to publicly reconstruct r from $\langle\langle r\rangle\rangle_t$.

4. Now P broadcasts the polynomials $F^l(x) = rfa^l(x) + f\beta^l(x)$ for $l = 1\ldots,\ell$ and $G^l(x) = p^l fb^l(x) - fd^l(x) - rfc^l(x)$ for $l = 1\ldots,\ell$, where $p^l = F^l(0)$.

5. Party $P_i \in \mathcal{P}$ checks whether $F^l(i) \stackrel{?}{=} rfa^l(i) + f\beta^l(i)$ for $l = 1,\ldots,\ell$. If the test fails for at least one l, then P_i raises a complaint and reveals $ICSig(P,P_i,\mathcal{P},(fa^1(i),\ldots,fa^\ell(i)))$ and $ICSig(P,P_i,\mathcal{P},(f\beta^1(i),\ldots,f\beta^\ell(i)))$ if the values are not public already. If a complaint is raised, then all parties publicly check whether $F^l(i) \stackrel{?}{=} rfa^l(i) + f\beta^l(i)$ for $l = 1,\ldots,\ell$, using the revealed values (revealed from the IC signatures) or the public values (in case they were public). If the test fails for at least one l, then every party concludes that P fails to prove $c = ab$ and protocol terminates here.

6. Now party $P_i \in \mathcal{P}$ checks whether $G^l(i) \stackrel{?}{=} p^l fb^l(i) - fd^l(i) - rfc^l(i)$ for $l = 1,\ldots,\ell$. If the test fails for at least one l, then P_i raises a complaint and reveals $ICSig(P,P_i,\mathcal{P},(fb^1(i),\ldots,fb^\ell(i)))$, $ICSig(P,P_i,\mathcal{P},(fc^1(i),\ldots,fc^\ell(i)))$ and $ICSig(P,P_i,\mathcal{P},(fd^1(i),\ldots,fd^\ell(i)))$ if the values are not public. If a complaint is raised, then all parties publicly check whether $G^l(i) \stackrel{?}{=} p^l fb^l(i) - fd^l(i) - rfc^l(i)$ for $l = 1,\ldots,\ell$, using the revealed values (revealed from the IC signatures) or the public values (in case they were public). If the test fails for at least one l, then every party concludes that P fails to prove $c = ab$ and protocol terminates here.

7. Every player checks whether $q^l = G^l(0) \stackrel{?}{=} 0$ for $l = 1,\ldots,\ell$. If the test fails for at least one l, then every party concludes that P fails to prove $c = ab$ and protocol terminates here. Otherwise P has proved that $c^l = a^l b^l$ for $l = 1,\ldots,\ell$.

if P shares $c^l \neq a^l b^l$ and/or $d^l \neq \beta^l b^l$, then the value $q^l = r(a^l b^l - c^l) + \beta^l b^l - d^l$ will be non-zero, except for only one value of r. But since r is randomly generated, the probability that r is that value is $\frac{1}{|\mathbb{F}|} \leq \frac{\epsilon}{n^4} \leq \epsilon$. Therefore if P has generated $\langle\langle(c^1, \ldots, c^\ell)\rangle\rangle_t^P$, then except with probability ϵ, $c^l = a^l b^l$ for $l = 1, \ldots, \ell$.

Secrecy: We now prove the secrecy of a^l, b^l, c^l for all $l = 1, \ldots, \ell$ when P is honest. From the secrecy property of 5VSS-MS-Share and property of $1d^{(\star,\ell)}$-sharing, a^l, b^l and c^l will remain secure. Now we will show that both p^l and q^l will not leak any information about a^l, b^l and c^l. Clearly $p^l = (ra^l + \beta^l)$ will look completely random to the adversary as β^l is randomly chosen. Furthermore $q^l = 0$ and hence q^l does not leak any information on a^l, b^l and c^l. Hence the lemma. □

Lemma 5.6 *Protocol ProveCeqAB achieves the following:*

1. **Round Complexity:** *Fifteen rounds.*
2. **Communication Complexity:** *Private and Broadcast communication of $\mathcal{O}((\ell n^2 + n^4)\log\frac{1}{\epsilon})$ bits.*

PROOF: Step 1 requires five rounds (invokes three instances of 5VSS-MS-Share in parallel). Step 3 requires seven rounds (invokes one instance of Random and then one instance of 5VSS-Rec). Step 4 requires one round. Step 5 and 6 can be executed in parallel (may invoke several instances of Reveal in parallel) and they require at most two rounds together. Step 7 involves only local computation. Hence in total ProveCeqAB requires at most fifteen rounds. The communication complexity can be verified easily. □

5.3.1.4 Robust Multiplication Protocol

We now finally present our protocol called Mult, which allows the parties to generate $\langle\langle a^1, \ldots, a^\ell\rangle\rangle_t, \langle\langle b^1, \ldots, b^\ell\rangle\rangle_t$ and $\langle\langle c^1, \ldots, c^\ell\rangle\rangle_t$, where a^l's and b^l's are random and $c^l = a^l b^l$ for $l = 1, \ldots, \ell$. Moreover, each one of the $2d^{(\star,\ell)}$-sharing will have $\frac{\epsilon}{n}$ error. For simplicity, we first explain the idea of the protocol to generate a single triple $\langle\langle a\rangle\rangle_t, \langle\langle b\rangle\rangle_t$ and $\langle\langle c\rangle\rangle_t$.

To generate random $\langle\langle a\rangle\rangle_t$ and $\langle\langle b\rangle\rangle_t$, we invoke two parallel executions of protocol Random with $\ell = 1$. We call these executions as Random$_a$ and Random$_b$ respectively. Before proceeding further, let us closely look into Random$_a$ and Random$_b$. In Random$_a$, each party P_i would have executed 5VSS-MS-Share as a dealer with $\ell = 1$ to generate $\langle\langle a^{P_i}\rangle\rangle_t^{P_i}$, where a^{P_i} is a random element from $\mathbb{F}$. Similarly, in Random$_b$, each party P_i would have executed 5VSS-MS-Share as a dealer with $\ell = 1$ to generate $\langle\langle b^{P_i}\rangle\rangle_t^{P_i}$, where b^{P_i} is a random element from $\mathbb{F}$. Recall that $Pass_a$ ($Pass_b$) denote the set of parties whose instance of 5VSS-MS-Share as a dealer is successful in Random$_a$ (Random$_b$). Thus everyone has computed $\langle\langle a\rangle\rangle_t = \sum_{P_i \in Pass_a}\langle\langle a^{P_i}\rangle\rangle_t^{P_i}$ and $\langle\langle b\rangle\rangle_t = \sum_{P_i \in Pass_b}\langle\langle b^{P_i}\rangle\rangle_t^{P_i}$. This implies that every party P_i holds $a_i = \sum_{P_j \in Pass_a} a_i^{P_j}$ and $b_i = \sum_{P_j \in Pass_b} b_i^{P_j}$. Here a_i and b_i are the i^{th} shares of a and b respectively, likewise $a_i^{P_j}$ and $b_i^{P_j}$ are i^{th} share of a^{P_j} and b^{P_j} respectively. Moreover, the parties hold $\langle a_i^{P_j}\rangle_t^{P_j}$ and $\langle b_i^{P_j}\rangle_t^{P_j}$, for $i = 1, \ldots, n$ (see section 4.3.1) for every P_j in $Pass_a$ and $Pass_b$ respectively. Furthermore, the parties hold $\langle a_i\rangle_t^{P_i}$ and $\langle b_i\rangle_t^{P_i}$ for $i = 1, \ldots, n$.

Now to generate $\langle\langle c \rangle\rangle_t$, we use the following idea from [48]: every party P_i computes $a_i b_i$ and generates $\langle\langle a_i b_i \rangle\rangle_t^{P_i}$ from $\langle a_i \rangle_t^{P_i}$ and $\langle b_i \rangle_t^{P_i}$, by executing ProveCeqAB. Notice that at most t corrupted parties may fail to generate $\langle\langle a_i b_i \rangle\rangle_t^{P_i}$. Since $a_1 b_1, \ldots, a_n b_n$ are n points on a $2t$ degree polynomial, say $C(x)$, whose constant term is c, by Lagrange interpolation formula [46], c can be computed as $c = \sum_{i=1}^{n} r_i(a_i b_i)$ where $r_i = \prod_{j=1, j\neq i}^{n} \frac{-j}{i-j}$. The vector $(r_1, \ldots, r_n)$ is called recombination vector [46] which is public and known to every party. So we write $c = Lagrange(a_1 b_1, \ldots, a_n b_n) = \sum_{i=1}^{n} r_i(a_i b_i)$. Now all parties can compute $\langle\langle c \rangle\rangle_t = Lagrange(\langle\langle a_1 b_1 \rangle\rangle_t^{P_1}, \ldots, \langle\langle a_n b_n \rangle\rangle_t^{P_n})$, to obtain the desired output. Notice that since $C(x)$ is of degree $2t$, we need all the P_i's to successfully generate $\langle\langle a_i b_i \rangle\rangle_t^{P_i}$ (a $2t$ degree polynomial requires $2t+1$ points on it to be interpolated correctly) in order to successfully generate $\langle\langle c \rangle\rangle_t$ using the above mechanism. Even if a single corrupted party P_i fails to generate $\langle\langle a_i b_i \rangle\rangle_t^{P_i}$, the above technique will fail. To make Mult robust, we reconstruct a_i and b_i publicly when P_i fails to generate $\langle\langle a_i b_i \rangle\rangle_t^{P_i}$ in ProveCeqAB. All the parties then proceeds with the above mentioned computation assuming $a_i b_i$ as a zero degree polynomial.

The a_i and b_i for a corrupted P_i who has failed to generate $\langle\langle a_i b_i \rangle\rangle_t^{P_i}$ in ProveCeqAB, can be publicly reconstructed as follows. As explained earlier, $a_i = \sum_{P_j \in Pass_a} a_i^{P_j}$ and $b_i = \sum_{P_j \in Pass_b} b_i^{P_j}$. Moreover, the parties hold $\langle a_i^{P_j} \rangle_t^{P_j}$ and $\langle b_i^{P_j} \rangle_t^{P_j}$. So we first try to publicly reconstruct $a_i^{P_j}$ and $b_i^{P_j}$ corresponding to every P_j in $Pass_a$ and $Pass_b$ respectively, using protocol Rec. From the properties of protocol Rec, corresponding to every *honest* P_j, the values $a_i^{P_j}$ and $b_i^{P_j}$ will be reconstructed correctly with very high probability. However, corresponding to a *corrupted* P_j, Rec may not output $a_i^{P_j}$ and $b_i^{P_j}$, in which case, everybody will come to know that P_j is corrupted. Like this, there can be at most t corrupted P_j's, corresponding to which the protocol Rec may fail to output $a_i^{P_j}$ and/or $b_i^{P_j}$. Let $\mathcal{C}$ be the set of such corrupted parties. Now corresponding to the parties in $\mathcal{C}$, everyone computes $\langle\langle \sum_{P_j \in \mathcal{C}} a^{P_j} \rangle\rangle_t, \langle\langle \sum_{P_j \in \mathcal{C}} b^{P_j} \rangle\rangle_t$ and use protocol 5VSS-Rec to publicly reconstruct $\sum_{P_j \in \mathcal{C}} a^{P_j}$ and $\sum_{P_j \in \mathcal{C}} b^{P_j}$. Once $\sum_{P_j \in \mathcal{C}} a^{P_j}$ and $\sum_{P_j \in \mathcal{C}} b^{P_j}$ are known publicly, the i^{th} shares of these values, namely $\sum_{P_j \in \mathcal{C}} a_i^{P_j}$ and $\sum_{P_j \in \mathcal{C}} b_i^{P_j}$ are also publicly known. Now everyone computes $a_i = \sum_{P_j \in (Pass_a \setminus \mathcal{C})} a_i^{P_j} + \sum_{P_j \in \mathcal{C}} a_i^{P_j}$ and $b_i = \sum_{P_j \in (Pass_b \setminus \mathcal{C})} b_i^{P_j} + \sum_{P_j \in \mathcal{C}} b_i^{P_j}$. Our protocol Mult (given in Fig. 5.4) follows the above ideas for ℓ pairs concurrently.

Mult works on a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^5 2^{-\kappa}$. This is because, Mult invokes ProveCeqAB with $\frac{\epsilon}{n}$ error probability.

Lemma 5.7 *Protocol Mult satisfies the following properties:*

1. **Correctness***: Except with probability ϵ, the protocol correctly outputs $(\langle\langle a^1, \ldots, a^\ell \rangle\rangle_t, \langle\langle b^1, \ldots, b^\ell \rangle\rangle_t, \langle\langle c^1, \ldots, c^\ell \rangle\rangle_t)$ where each of the three $2d^{(\star,\ell)}$-sharing will have $\frac{\epsilon}{n}$ error. Moreover, for $l = 1, \ldots, \ell$, $c^l = a^l b^l$.*

2. **Secrecy***: The adversary will have no information about (a^k, b^k, c^k), for $k = 1, \ldots, \ell$.*

PROOF: **Correctness:** First of all, if at all $\langle\langle a^1, \ldots, a^\ell \rangle\rangle_t$, $\langle\langle b^1, \ldots, b^\ell \rangle\rangle_t$ and $\langle\langle c^1, \ldots, c^\ell \rangle\rangle_t$ are generated, they will have $\frac{\epsilon}{n}$ error. This follows from the **Correctness** of protocol Random and the fact that $\langle\langle c^1, \ldots, c^\ell \rangle\rangle_t$ is computed as the linear combination of at least $t+1$ $2d^{(\star,\ell)}$-sharing, each having $\frac{\epsilon}{n}$ error.

Figure 5.4: Robust Multiplication Protocol.

$$(\langle\langle a^1,\ldots,a^\ell\rangle\rangle_t, \langle\langle b^1,\ldots,b^\ell\rangle\rangle_t, \langle\langle c^1,\ldots,c^\ell\rangle\rangle_t) = \mathsf{Mult}(\mathcal{P},\ell,\epsilon)$$

1. The parties invoke $\mathsf{Random}(\mathcal{P},\ell,\epsilon)$ twice in parallel to generate $\langle\langle a^1,\ldots,a^\ell\rangle\rangle_t$ and $\langle\langle b^1,\ldots,b^\ell\rangle\rangle_t$, each having $\frac{\epsilon}{n}$ error. These two executions are denoted by Random_a and Random_b.
2. Let $Pass_a$ ($Pass_b$) denote the set of parties whose instances of 5VSS-MS-Share as a dealer are successful in Random_a (Random_b). Then we have
   (a) In Random_a, $P_i \in Pass_a$ had generated $\langle\langle a^{1i},\ldots,a^{\ell i}\rangle\rangle_t^{P_i}$ having $\frac{\epsilon}{n}$ error.
   (b) Similarly, in Random_b, $P_i \in Pass_b$ had generated $\langle\langle b^{1i},\ldots,b^{\ell i}\rangle\rangle_t^{P_i}$ having $\frac{\epsilon}{n}$ error.
   (c) According to the steps of Random, $a^l = \sum_{P_j\in Pass_a} a^{lj}$ and $b^l = \sum_{P_j\in Pass_b} b^{lj}$, for $l = 1,\ldots,\ell$.
   (d) Let a_i^l and b_i^l denote the i^{th} share of a^l and b^l respectively. Clearly $a_i^l = \sum_{P_j\in Pass_a} a_i^{lj}$ and $b_i^l = \sum_{P_j\in Pass_b} b_i^{lj}$, where a_i^{lj} and b_i^{lj} are i^{th} shares of a^{lj} and b^{lj}.
   (e) $\langle a_i^1,\ldots,a_i^\ell\rangle_t^{P_i}$ and $\langle b_i^1,\ldots,b_i^\ell\rangle_t^{P_i}$ are available for all honest P_i's and each of these sharing has $\frac{\epsilon}{n^2}$ error.
3. Party P_i invokes $\mathsf{ProveCeqAB}(P_i,\mathcal{P},\langle a_i^1,\ldots,a_i^\ell\rangle_t^{P_i},\langle b_i^1,\ldots,b_i^\ell\rangle_t^{P_i},\frac{\epsilon}{n})$ to generate $\langle\langle c_i^1,\ldots,c_i^\ell\rangle\rangle_t^{P_i}$ having $\frac{\epsilon}{n}$ error.
4. Let $Fail$ be the set of parties P_i who failed during their instance of ProveCeqAB. We reconstruct $a_i^1,\ldots,a_i^\ell$ and $b_i^1,\ldots,b_i^\ell$ publicly for every $P_i \in Fail$ by executing the following steps. We describe the steps with respect to $a_i^1,\ldots,a_i^\ell$ only. Similar steps should be executed for $b_i^1,\ldots,b_i^\ell$.
   (a) For every $P_i \in Fail$, we first try to reconstruct $a_i^{1j},\ldots,a_i^{\ell j}$ corresponding to each $P_j \in Pass_a$ from $\langle a_i^{1j},\ldots,a_i^{\ell j}\rangle_t^{P_j}$, having $\frac{\epsilon}{n^2}$ error, which is generated by P_j in Random_a.
      i. For each $P_j \in Pass_a$, the parties execute $\mathsf{Rec}(\mathcal{P},\ell,\langle a_i^{1j},\ldots,a_i^{\ell j}\rangle_t^{P_j},\frac{\epsilon}{n^2})$ to either publicly reconstruct $a_i^{1j},\ldots,a_i^{\ell j}$ or detect P_j as corrupted.
      ii. Let $\mathcal{C}_i$ denote the set of all P_j's who are detected to be corrupted in their respective instance of $\mathsf{Rec}(\mathcal{P},\ell,\langle a_i^{1j},\ldots,a_i^{\ell j}\rangle_t^{P_j},\frac{\epsilon}{n^2})$.
   (b) Let $\mathcal{C} = \cup_{P_i\in Fail}\mathcal{C}_i$.
   (c) The parties execute $\mathsf{5VSS\text{-}MS\text{-}Rec}(\mathcal{P},\langle\langle\sum_{P_j\in\mathcal{C}} a^{1j},\ldots,\sum_{P_j\in\mathcal{C}} a^{\ell j}\rangle\rangle_t,\frac{\epsilon}{n})$ to publicly reconstruct $\sum_{P_j\in\mathcal{C}} a^{1j},\ldots,\sum_{P_j\in\mathcal{C}} a^{\ell j}$ (the probability used in argument of 5VSS-MS-Rec comes from the fact that each of $\langle\langle a^{1j},\ldots,a^{\ell j}\rangle\rangle_t^{P_j}$ has $\frac{\epsilon}{n}$ error; thus $\langle\langle\sum_{P_j\in\mathcal{C}} a^{1j},\ldots,\sum_{P_j\in\mathcal{C}} a^{\ell j}\rangle\rangle_t$ will have $\frac{\epsilon}{n}$ error).
   (d) Every party computes $a_i^l = \sum_{P_j\in\mathcal{C}} a_i^{lj} + \sum_{P_j\in(Pass_a\setminus\mathcal{C})} a_i^{lj}$ for every $P_i \in Fail$.
5. Every party finds $c_i^l = a_i^l b_i^l$ for every $P_i \in Fail$.
6. All the parties compute: $\langle\langle c^1,\ldots,c^\ell\rangle\rangle_t = \sum_{(\mathcal{P}\setminus Fail)} r_i\langle\langle c_i^1,\ldots,c_i^\ell\rangle\rangle_t^{P_i} + \sum_{P_i\in Fail} r_i(c_i^1,\ldots,c_i^\ell)$, where $(r_1,\ldots,r_{2t+1})$ represents the recombination vector [46].

We now show that Mult will be able to generate its output except with probability ϵ. To assert the error probability of Mult, we compute and show that the error probability of the following two events are ϵ:

1. **All the parties in $\mathcal{P}\setminus Fail$ has generated correct $\langle\langle c_i^1,\ldots,c_i^\ell\rangle\rangle_t^{P_i}$:** We show that this event has an error probability of ϵ. Every P_i who is successful

in $\mathsf{ProveCeqAB}$ will generate $\langle\langle c_i^1, \ldots, c_i^\ell \rangle\rangle_t^{P_i}$, except with error probability $\frac{\epsilon}{n}$ (see Lemma 5.5). Since $|\mathcal{P} \setminus Fail| \geq t+1$ (all the $t+1$ honest P_i's will be present in $\mathcal{P} \setminus Fail$, except with error probability ϵ; follows from Lemma 5.5), all the parties in $\mathcal{P} \setminus Fail$ has generated correct $\langle\langle c_i^1, \ldots, c_i^\ell \rangle\rangle_t^{P_i}$, except with probability $(t+1)\frac{\epsilon}{n} \approx \epsilon$. Hence $\sum_{P_i \in (\mathcal{P} \setminus Fail)} r_i \langle\langle c_i^1, \ldots, c_i^\ell \rangle\rangle_t^{P_i}$ will be generated correctly, except with probability ϵ.

2. **For every party P_i in $Fail$, $a_i^1, \ldots, a_i^\ell$ and $b_i^1, \ldots, b_i^\ell$ will be reconstructed correctly:** We now show that this event also has an error probability ϵ. Consequently, it implies that $\sum_{P_i \in Fail} r_i(c_i^1, \ldots, c_i^\ell)$ will be generated correctly except with probability ϵ. This case together with the previous case will complete our proof.

   First we compute the error probability involved in reconstructing $a_i^1, \ldots, a_i^\ell$, corresponding to all $P_i \in Fail$. Same argument will follow for $b_i^1, \ldots, b_i^\ell$. There are two events to be accounted here (first event has an error probability ϵ and the second event has error probability $\frac{\epsilon}{n}$):

   (a) *The values $a_i^{1j}, \ldots, a_i^{\ell j}$ corresponding to all $P_j \in Pass_a \setminus \mathcal{C}$ and all $P_i \in Fail$ will be reconstructed correctly:* Recall that for reconstructing $a_i^1, \ldots, a_i^\ell$, we tried to reconstruct $a_i^{1j}, \ldots, a_i^{\ell j}$ corresponding to each $P_j \in Pass_a$ from $\langle a_i^{1j}, \ldots, a_i^{\ell j} \rangle_t^{P_j}$ generated by P_j in Random_a. Notice that since $\langle\langle a^{1j}, \ldots, a^{\ell j} \rangle\rangle_t^{P_j}$ has $\frac{\epsilon}{n}$ error, it implies that $\langle a_i^{1j}, \ldots, a_i^{\ell j} \rangle_t^{P_j}$ will have $\frac{\epsilon}{n^2}$ error. Now since $|Pass_a \setminus \mathcal{C}| \geq t+1$ (because all honest parties in $Pass_a$ will also be present in $Pass_a \setminus \mathcal{C}$ with very high probability), the values $a_i^{1j}, \ldots, a_i^{\ell j}$ corresponding to all $P_j \in Pass_a \setminus \mathcal{C}$ will be reconstructed correctly, except with probability $(t+1)\frac{\epsilon}{n^2} \approx \frac{\epsilon}{n}$ (follows from Lemma 5.3).

   Now since there can be at most t P_i's in $Fail$, it implies that except with probability $t\frac{\epsilon}{n} \approx \epsilon$, the values $a_i^{1j}, \ldots, a_i^{\ell j}$ corresponding to all P_j's in $Pass_a \setminus \mathcal{C}$ and all P_i's in $Fail$ will be reconstructed correctly.

   (b) *The values $\sum_{P_j \in \mathcal{C}} a_i^{1j}, \ldots, \sum_{P_j \in \mathcal{C}} a_i^{\ell j}$ for all $P_i \in Fail$ will be reconstructed correctly:* Recall that the parties execute $\mathsf{5VSS\text{-}MS\text{-}Rec}(\mathcal{P}, \langle\langle \sum_{P_j \in \mathcal{C}} a^{1j}, \ldots, \sum_{P_j \in \mathcal{C}} a^{\ell j} \rangle\rangle_t, \frac{\epsilon}{n})$ to publicly reconstruct $\sum_{P_j \in \mathcal{C}} a^{1j}, \ldots, \sum_{P_j \in \mathcal{C}} a^{\ell j}$. The reconstruction will be successful, except with error probability $\frac{\epsilon}{n}$. This follows from the fact that each of $\langle\langle a^{1j}, \ldots, a^{\ell j} \rangle\rangle_t^{P_j}$ are generated in $\mathsf{5VSS\text{-}MS\text{-}Share}(P_j, \mathcal{P}, (a^{1j}, \ldots, a^{\ell j} \rangle\rangle_t, \frac{\epsilon}{n})$ with error parameter $\frac{\epsilon}{n}$ and therefore $\langle\langle \sum_{P_j \in \mathcal{C}} a^{1j}, \ldots, \sum_{P_j \in \mathcal{C}} a^{\ell j} \rangle\rangle_t$ will have $\frac{\epsilon}{n}$ error. Thus $\sum_{P_j \in \mathcal{C}} a^{1j}, \ldots, \sum_{P_j \in \mathcal{C}} a^{\ell j}$ will be reconstructed correctly except with error probability $\frac{\epsilon}{n}$.

Once $\sum_{P_j \in \mathcal{C}} a^{1j}, \ldots, \sum_{P_j \in \mathcal{C}} a^{\ell j}$ and all $a_i^{1j}, \ldots, a_i^{\ell j}$ corresponding to all $P_j \in Pass_a \setminus \mathcal{C}$ are publicly known, the parties can publicly reconstruct $a_i^1, \ldots, a_i^\ell$ corresponding to each $P_i \in Fail$. Thus parties can reconstruct $\sum_{P_i \in Fail} r_i(c_i^1, \ldots, c_i^\ell)$ correctly, except with probability $\epsilon + \frac{\epsilon}{n} \approx \epsilon$.

Now as $\langle\langle c^1, \ldots, c^\ell \rangle\rangle_t = \sum_{P_i \in (\mathcal{P} \setminus Fail)} r_i \langle\langle c_i^1, \ldots, c_i^\ell \rangle\rangle_t^{P_i} + \sum_{P_i \in Fail} r_i(c_i^1, \ldots, c_i^\ell)$, the error probability in generating the above is $\epsilon + \epsilon \approx \epsilon$.

Secrecy: From the secrecy property of Random, the values $(a_i^1, \ldots, a_i^\ell)$ and $(b_i^1, \ldots, b_i^\ell)$ are completely random and unknown to $\mathcal{A}_t$. Now According to the secrecy of protocol $\mathsf{ProveCeqAB}$, $(c_i^1, \ldots, c_i^\ell)$, $(a_i^1, \ldots, a_i^\ell)$ and $(b_i^1, \ldots, b_i^\ell)$ will remain secure for every honest P_i. Now if a corrupted P_i fails to generate $\langle\langle c_i^1, \ldots, c_i^\ell \rangle\rangle_t^{P_i}$ in $\mathsf{ProveCeqAB}$, then we reconstruct $a_i^1, \ldots, a_i^\ell$ and $b_i^1, \ldots, b_i^\ell$ which are already known to adversary. In fact all the values that are publicly reconstructed in the protocol are already known to adversary. Now as $\langle\langle c^1, \ldots, c^\ell \rangle\rangle_t$ is generated by taking linear combination of $\langle\langle c_i^1, \ldots, c_i^\ell \rangle\rangle_t^{P_i}$'s (in which at least $t+1$ set of $c_i^1, \ldots, c_i^\ell$ are unknown to $\mathcal{A}_t$), the secrecy of $c^1, \ldots, c^\ell$ is guaranteed. □

Lemma 5.8 *Protocol Mult has the following bounds:*

1. **Round Complexity:** *Twenty four rounds.*
2. **Communication Complexity:** *Private and broadcast communication of $\mathcal{O}((\ell n^3 + n^5) \log \frac{1}{\epsilon})$ bits.*

PROOF: Step 1 requires five rounds (invokes two instances of Random in parallel). Step 3 requires fifteen rounds (invokes n parallel instances of $\mathsf{ProveCeqAB}$). Step 4 requires four rounds (invokes several instances of Rec in parallel and then several instances of $\mathsf{5VSS\text{-}MS\text{-}Rec}$ in parallel). In total Mult requires twenty four rounds. The communication complexity of Mult can be verified easily. □

5.3.2 Conversion From $2d^{(\star,\ell)}$-sharing to ℓ Individual $2d^\star$-sharing

In the previous section, we have generated $2d^{(\star,\ell)}$-sharing of random multiplication triples, namely $\langle\langle a^1, \ldots, a^\ell \rangle\rangle_t$, $\langle\langle b^1, \ldots, b^\ell \rangle\rangle_t$ and $\langle\langle c^1, \ldots, c^\ell \rangle\rangle_t$, each having $\frac{\epsilon}{n}$ error. But recall that the goal of Preparation phase was to generate $2d^\star$-sharing of each of the a, b and c values, namely $(\langle\langle a^l \rangle\rangle_t, \langle\langle b^l \rangle\rangle_t, \langle\langle c^l \rangle\rangle_t)$, where $l = 1, \ldots, c_M + c_R$, with each $2d^\star$-sharing having $\frac{\epsilon}{(2c_M + c_O)}$ error. It is to be noted that there are two issues that need to be handled to attain the goal of Preparation phase: (a) generation of $2d^\star$-sharing of individual secrets from $2d^{(\star,\ell)}$-sharing; (b) change of error i.e given $2d^{(\star,\ell)}$-sharing with $\frac{\epsilon}{n}$ error, we have to generate $2d^\star$-sharings having $\frac{\epsilon}{(2c_M + c_O)}$ error.

The first issue is necessary to handle due to the following reason: In preparation phase, generating sharing of individual values is essential as otherwise due to the inherent limitation of $2d^{(\star,\ell)}$-sharing (as mentioned in subsection 4.3.1), an attempt to reconstruct a single value out of those ℓ secrets will reveal the entire set of values. This is certainly undesirable as this may lead to breach of secrecy in the following way: recall that we stated in the overview of our statistical MPC that one multiplication triple (each value is $2d^\star$-shared) will be associated with each multiplication gate of the circuit and the gate will be computed with the help of it using two reconstructions; Now if we associate the multiplication triple while they are still $2d^{(\star,\ell)}$-shared, reconstruction for the first multiplication gate will disclose all other multiplication triples. Thus the separation of the secrets are necessary.

The second issue has to be ensured to bound the error probability of Computation phase by ϵ (details are provided in section 5.5). Hence at this juncture, we require a technique to convert a $2d^{(\star,\ell)}$-sharing of ℓ values having error ϵ to ℓ separate $2d^\star$-sharing of same values having error δ, while maintaining the secrecy of those values. In this section, we attempt the same and devise a protocol called

$\mathsf{Convert}$ to achieve the above goal. Assuming that the given $2d^{(\star,\ell)}$-sharing has ϵ error, protocol $\mathsf{Convert}$ generates each of the individual $2d^{\star}$-sharing having δ error.

Protocol $\mathsf{Convert}$ works on a field $\mathbb{F} = GF(2^{\kappa})$, where κ has to be determined using relation $\epsilon \geq n^3 2^{-\kappa}$ as well as $\delta \geq n^3 2^{-\kappa}$. This is because, $\mathsf{Convert}$ uses $\mathsf{MVMS\text{-}ICP}$ with $\frac{\epsilon}{n^2}$ as well as $\frac{\delta}{n^2}$ error probability. So here $\kappa = \mathcal{O}(\log \frac{1}{\max(\epsilon,\delta)})$. The protocol is formally given in Fig. 5.5.

Figure 5.5: Protocol for converting $2d^{(\star,\ell)}$-sharing to ℓ separate $2d^{\star}$-sharing.

$$(\langle\langle s^1\rangle\rangle_t, \ldots, \langle\langle s^{\ell}\rangle\rangle_t) = \textbf{Convert}(\mathcal{P}, \ell, \langle\langle s^1, \ldots, s^{\ell}\rangle\rangle_t, \epsilon, \delta)$$

Let $S_i = (s_i^1, \ldots, s_i^{\ell})$ be the i^{th} shares of $S = (s^1, \ldots, s^{\ell})$ and $S_{ij} = (s_{ij}^1, \ldots, s_{ij}^{\ell})$ be the j^{th} share-share of $S_i = (s_i^1, \ldots, s_i^{\ell})$. Now given $\langle\langle s^1, \ldots, s^{\ell}\rangle\rangle_t$ having ϵ error, it implies that corresponding to honest P_i, either S_{ij} is publicly known or honest P_j holds $ICSig(P_i, P_j, \mathcal{P}, S_{ij})$ having $\frac{\epsilon}{n^2}$ error. So for every pair of parties (P_i, P_j), party P_i and P_j do the following communication:

1. If S_{ij} is known in public, then parties P_i and P_j do not communicate anything.
2. Otherwise, P_i sends $ICSig(P_i, P_j, \mathcal{P}, s_{ij}^l)$ having $\frac{\delta}{n^2}$ error to P_j for all $l \in \{1, \ldots, \ell\}$. Upon receiving $ICSig(P_i, P_j, \mathcal{P}, \overline{s_{ij}^l})$ having $\frac{\delta}{n^2}$ error from P_i for all $l \in \{1, \ldots, \ell\}$, party P_j now checks if $\overline{s_{ij}^l} = s_{ij}^l$ for all $l \in \{1, \ldots, \ell\}$.
3. For every l for which the above test fails, P_j reveals $ICSig(P_i, P_j, \mathcal{P}, \overline{s_{ij}^l})$ having $\frac{\delta}{n^2}$ error. P_j also reveals $ICSig(P_i, P_j, \mathcal{P}, S_{ij})$ having $\frac{\epsilon}{n^2}$ error.
4. If P_j has successfully revealed $ICSig(P_i, P_j, \mathcal{P}, \overline{s_{ij}^l})$ and $ICSig(P_i, P_j, \mathcal{P}, S_{ij})$ and indeed there is a mismatch for l^{th} value, then all parties ignore all the information received so far from P_i regarding $\langle\langle s^l\rangle\rangle_t$.

Lemma 5.9 *Protocol $\mathsf{Convert}$ achieves the following properties:*

1. **Correctness:** *Given $\langle\langle s^1, \ldots, s^{\ell}\rangle\rangle_t$ having ϵ error, the protocol produces $\langle\langle s^l\rangle\rangle_t$ having δ error for $l = 1, \ldots, \ell$, except with probability* $\max(\epsilon, \delta)$.
2. **Secrecy:** *The values $s^1, \ldots, s^{\ell}$ remain secure.*

PROOF: **Correctness:** It is clear that if at all $\langle\langle s^l\rangle\rangle_t$ is generated, then it will have δ error, as each of its underlying IC signature, namely $ICSig(P_i, P_j, \mathcal{P}, s_{ij}^l)$ will have $\frac{\delta}{n^2}$ error.

Next we show that protocol $\mathsf{Convert}$ will generate ℓ $2d^{\star}$-sharing, except with probability $\max(\epsilon, \delta)$. First notice that for every honest pair (P_i, P_j), P_j will correctly receive $ICSig(P_i, P_j, \mathcal{P}, \overline{s_{ij}^l})$ with $\overline{s_{ij}^l} = s_{ij}^l$ for all $l \in \{1, \ldots, \ell\}$. Now for an $l \in \{1, \ldots, \ell\}$, the sharing $\langle\langle s^l\rangle\rangle_t$ will not be generated if some corrupted P_j is able to accuse some honest P_i by revealing $ICSig(P_i, P_j, \mathcal{P}, \overline{s_{ij}^l})$ as well as $ICSig(P_i, P_j, \mathcal{P}, S_{ij})$ such that $\overline{s_{ij}^l} \neq s_{ij}^l$. In this case P_i's signatures with respect to $\langle\langle s^l \rangle\rangle_t$ will be ignored by *everybody*. This will violate definition of $2d^\star$-sharing as it demands that corresponding to every honest P_i, every other honest P_j should hold $ICSig(P_i, P_j, \mathcal{P}, s^l_{ij})$ or the value s^l_{ij} should be public. But a corrupted P_j can accuse honest P_i and make every body ignore P_i's information for some $l \in \{1, \ldots, \ell\}$ with probability only $\max(\frac{\epsilon}{n^2}, \frac{\delta}{n^2})$, as P_j can forge $ICSig(P_i, P_j, \mathcal{P}, \overline{s^l_{ij}})$ with probability $\frac{\delta}{n^2}$ or $ICSig(P_i, P_j, \mathcal{P}, S_{ij})$ with probability $\frac{\epsilon}{n^2}$. Now as there can be $(t+1)t$ pairs with one honest and one corrupted party, the probability that some honest party is accused by some corrupted party is at most $\max(\epsilon, \delta)$.

Secrecy: Secrecy follows from the secrecy of protocol MVMS-ICP and due to the fact that the revealed values are already known to adversary. □

Lemma 5.10 *Protocol Convert has the following bounds:*

1. **Round Complexity:** *Five Rounds.*
2. **Communication Complexity:** *Private and broadcast communication of* $\mathcal{O}(\ell n^3 \log \frac{1}{\max(\delta, \epsilon)})$ *bits.*

Proof: Convert executes several instances of Gen, Ver and Reveal in the sequence. Therefore, it requires five rounds. Communication complexity is easy to verify. □

5.3.3 Preparation Phase — Main Protocol

Here we present the protocol for preparation phase (called as PreparationPhase) where $2d^\star$-sharing of $c_M + c_R$ random multiplication triples $((a^k, b^k, c^k)\ ;\ k = 1, \ldots, c_M + c_R)$ are generated, each having $\frac{\epsilon}{(2c_M + c_O)}$ error.

PreparationPhase works on a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^3 2^{-\kappa} \cdot \max(n^2, (2c_M + c_O))$. Since PreparationPhase invokes Mult with ϵ error probability, $\epsilon \geq n^5 2^{-\kappa}$ should hold. Similarly, since PreparationPhase invokes Convert with $\frac{\epsilon}{n}$ and $\frac{\epsilon}{(2c_M + c_O)}$ error probability, $\epsilon \geq n^4 2^{-\kappa}$ as well as $\epsilon \geq (2c_M + c_O) n^3 2^{-\kappa}$ should hold. Therefore, $\epsilon \geq \max(n^5 2^{-\kappa}, (2c_M + c_O) n^3 2^{\kappa}) = n^3 2^{-\kappa} \cdot \max(n^2, (2c_M + c_O))$. Here $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$, because of $n = \mathcal{O}(\log \frac{1}{\epsilon})$ and the assumption $n = \mathrm{poly}(c_M, c_O, c_A, c_I)$.

Lemma 5.11 *Except with error probability of ϵ, protocol PreparationPhase produces correct $2d^\star$-sharing of $(c_M + c_R)$ secret multiplication triples, where each sharing has $\frac{\epsilon}{(2c_M + c_O)}$ error.*

Proof: The error probability of ϵ of protocol PreparationPhase comes from the executions of Mult and Convert. The output sharings will have $\frac{\epsilon}{(2c_M + c_O)}$ error. This follows from the **Correctness** of Convert. □

Lemma 5.12 *Protocol PreparationPhase achieves the following:*

1. **Round Complexity:** *Twenty Nine Rounds.*
2. **Communication Complexity:** *Private and broadcast communication of* $\mathcal{O}((c_M + c_R) n^3 + n^5) \log \frac{1}{\epsilon})$ *bits.*

Figure 5.6: Protocol for generating $2d^\star$-sharing of $c_M + c_R$ random multiplication triples $((a^l, b^l, c^l)\ ;\ l = 1, \ldots, c_M + c_R)$.

$$\{(\langle\langle a^l\rangle\rangle_t, \langle\langle b^l\rangle\rangle_t, \langle\langle c^l\rangle\rangle_t); l = 1, \ldots, c_M + c_R\} \;=\; \textsf{PreparationPhase}(\mathcal{P}, \epsilon)$$

1. Parties execute protocol $\textsf{Mult}(\mathcal{P}, c_M + c_R, \epsilon)$ to generate $\langle\langle a^1, \ldots, a^{c_M+c_R}\rangle\rangle_t, \langle\langle b^1, \ldots, b^{c_M+c_R}\rangle\rangle_t, \langle\langle c^1, \ldots, c^{c_M+c_R}\rangle\rangle_t$, each having $\frac{\epsilon}{n}$ error.

2. Parties execute

   (a) $\textsf{Convert}(\mathcal{P}, c_M + c_R, \langle\langle a^1, \ldots, a^{c_M+c_R}\rangle\rangle_t, \frac{\epsilon}{n}, \frac{\epsilon}{(2c_M+c_O)})$, to generate $(\langle\langle a^1\rangle\rangle_t, \ldots, \langle\langle a^{c_M+c_R}\rangle\rangle_t)$, each having $\frac{\epsilon}{(2c_M+c_O)}$ error;

   (b) $\textsf{Convert}(\mathcal{P}, c_M + c_R, \langle\langle b^1, \ldots, b^{c_M+c_R}\rangle\rangle_t, \frac{\epsilon}{n}, \frac{\epsilon}{(2c_M+c_O)})$, to generate $(\langle\langle b^1\rangle\rangle_t, \ldots, \langle\langle b^{c_M+c_R}\rangle\rangle_t)$, each having $\frac{\epsilon}{(2c_M+c_O)}$ error and

   (c) $\textsf{Convert}(\mathcal{P}, c_M + c_R, \langle\langle c^1, \ldots, c^{c_M+c_R}\rangle\rangle_t, \frac{\epsilon}{n}, \frac{\epsilon}{(2c_M+c_O)})$, to generate $(\langle\langle c^1\rangle\rangle_t, \ldots, \langle\langle c^{c_M+c_R}\rangle\rangle_t)$, each having $\frac{\epsilon}{(2c_M+c_O)}$ error.

3. All the parties output $(\langle\langle a^l\rangle\rangle_t, \langle\langle a^l\rangle\rangle_t, \langle\langle c^l\rangle\rangle_t); l = 1, \ldots, c_M + c_R$, each having $\frac{\epsilon}{(2c_M+c_O)}$ error.

PROOF: Step 1 requires twenty four rounds (invokes one instance of Mult). Step 2 requires five rounds (invokes three instances of Convert in parallel). Therefore the round complexity of PreparationPhase is twenty nine rounds. The communication complexity of PreparationPhase can be verified easily. □

5.4 Input Phase

The goal of the Input Phase is to generate $2d^\star$-sharing of the inputs of each party, where every sharing has $\frac{\epsilon}{\max(n^2,(2c_M+c_O))}$ error. Assume that $P_i \in \mathcal{P}$ has c_i inputs with $c_i = \mathcal{O}(n)$. So total number of input gates $c_I = \sum_{i=1}^n c_i$. Now in Input Phase, each P_i on having inputs $s^{i1}, \ldots, s^{ic_i}$, execute $\textsf{5VSS-Share}(P_i, \mathcal{P}, s^{il}, \frac{\epsilon}{\max(n^2,(2c_M+c_O))})$ for all $l = 1, \ldots, c_i$. If P_i is discarded during $\textsf{5VSS-Share}(P_i, \mathcal{P}, s^{il}, \frac{\epsilon}{\max(n^2,(2c_M+c_O))})$, then everyone assumes c_i predefined values on behalf of P_i.

InputPhase works on a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^3 2^{-\kappa} \cdot \max(n^2, (2c_M + c_O))$. This is the same field that PreparationPhase worked on. Since InputPhase invokes 5VSS-Share with $\frac{\epsilon}{\max(n^2,(2c_M+c_O))}$ error probability, we require $\epsilon \geq n^3 2^{-\kappa} \cdot \max(n^2, (2c_M + c_O))$. The protocol is given in Fig. 5.7.

Lemma 5.13 *Except with error probability of at most ϵ, protocol InputPhase produces correct $2d^\star$-sharing of all the inputs of the honest parties, where each sharing will have $\frac{\epsilon}{\max(n^2,(2c_M+c_O))}$ error.*

Proof: Every honest party will generate $2d^\star$-sharing of all its inputs, except with error probability $\frac{\epsilon c_i}{\max(n^2,(2c_M+c_O))}$. Since there are at least $t + 1$ honest parties, all

Figure 5.7: Protocol for generating $2d^\star$-sharing of the inputs of each party.

$$(\langle\langle s^{i1}\rangle\rangle_t, \ldots, \langle\langle s^{ic_i}\rangle\rangle_t; i = 1, \ldots, n) = \textsf{InputPhase}(\mathcal{P}, \epsilon)$$

1. Every party P_i on having inputs $s^{i1}, \ldots, s^{ic_i}$, execute 5VSS-Share$(P_i, \mathcal{P}, s^{il}, \frac{\epsilon}{\max(n^2,(2c_M+c_O))})$ for all $l = 1, \ldots, c_i$.
2. If P_i is discarded during 5VSS-Share$(P_i, \mathcal{P}, s^{il}, \frac{\epsilon}{\max(n^2,(2c_M+c_O))})$, then everyone assumes c_i predefined values on behalf of P_i.

of them will generate $2d^\star$-sharing of all their inputs, except with error probability $\frac{\epsilon c_H}{\max(n^2,(2c_M+c_O))}$, where c_H is the sum of c_i's corresponding to honest parties. Now we have $\frac{\epsilon c_H}{\max(n^2,(2c_M+c_O))} \approx \epsilon$ since $c_i = \mathcal{O}(n)$ and thus $c_H = \mathcal{O}(n^2)$. □

Lemma 5.14 *Protocol InputPhase has the following bounds:*

1. **Round Complexity:** *Five Rounds.*
2. **Communication Complexity:** *Private and broadcast communication of* $\mathcal{O}((c_I n^3 + n^5)\log\frac{1}{\epsilon})$ *bits.*

5.5 Computation Phase

Once Preparation Phase and Input Phase are over, the computation of the circuit (of the agreed upon function f) proceeds gate-by-gate. First, to every random and every multiplication gate, a prepared $2d^\star$-shared random multiplication triple (generated during **Preparation Phase**) is assigned. A gate (except output gate) g is said to be *evaluated* if a $2d^\star$-sharing $\langle\langle x\rangle\rangle_t$ is computed for the gate using the $2d^\star$-sharing of the inputs of the gate. Note that all the random and input gates will be *evaluated* as soon as we assign $2d^\star$-shared random triples (generated in **Preparation Phase**) and $2d^\star$-shared inputs (generated in **Input Phase**) to them respectively. A gate is said to be in *ready state*, when all its input gates have been *evaluated*. In the **Computation Phase**, the circuit evaluation proceeds in rounds where in each round all the ready gates will be evaluated in parallel. Evaluation of input and random gates do not require any communication. Due to linearity of $2d^\star$-sharing, linear gates can be evaluated without any communication.

For evaluating a multiplication gate, we use Beaver's *Circuit Randomization* technique [5]. Let x and y be the inputs of a multiplication gate, such that parties hold $\langle\langle x\rangle\rangle_t$ and $\langle\langle y\rangle\rangle_t$. Moreover, let $(\langle\langle a\rangle\rangle_t, \langle\langle b\rangle\rangle_t, \langle\langle c\rangle\rangle_t)$ be the multiplication triple (generated during **Preparation Phase**), which is associated with the multiplication gate. Now the parties want to generate $\langle\langle z\rangle\rangle_t$, where $z = xy$. Moreover, if x and y are unknown to $\mathcal{A}_t$, then x, y and z should be still unknown to $\mathcal{A}_t$. This can be done using Beaver's *Circuit Randomization* technique as follows: xy can be written as $xy = ((x-a)+a)((y-b)+b)$. Let $\alpha = (x-a)$ and $\beta = (y-b)$. The parties compute $\langle\langle \alpha\rangle\rangle_t$ and $\langle\langle \beta\rangle\rangle_t$. Then the parties reconstruct α and β. For this the parties execute protocol 5VSS-Rec. Once α and β are known to everyone, the parties compute $\langle\langle z\rangle\rangle_t = \alpha\beta + \alpha\langle\langle b\rangle\rangle_t + \beta\langle\langle a\rangle\rangle_t + \langle\langle c\rangle\rangle_t$.

The secrecy of x, y and z follows from the fact a, b are completely random and unknown to $\mathcal{A}_t$ [5]. As soon as an output gate becomes ready, the input to the output gate is reconstructed by every party by executing protocol 5VSS-Rec. The protocol for **Computation Phase** is given in Fig. 5.8. This protocol works on the same field that was used by both PreparationPhase and InputPhase.

Figure 5.8: Protocol for computing the circuit.

ComputationPhase$(\mathcal{P}, \epsilon)$

If a gate is in *ready state*, *compute* the gate in the following way depending on the type of the gate:

Input Gate: $\langle\langle s\rangle\rangle_t = \mathsf{IGate}(\langle\langle s\rangle\rangle_t)$

1. No computation is performed here. Simply output $\langle\langle s\rangle\rangle_t$.

Random Gate: $\langle\langle a\rangle\rangle_t = \mathsf{RGate}(\langle\langle a\rangle\rangle_t, \langle\langle b\rangle\rangle_t, \langle\langle c\rangle\rangle_t)$

1. No computation is performed here. Simply output $\langle\langle a\rangle\rangle_t$.

Addition Gate: $\langle\langle z\rangle\rangle_t = \mathsf{AGate}(\langle\langle x\rangle\rangle_t, \langle\langle y\rangle\rangle_t)$

1. Compute and output $\langle\langle z\rangle\rangle_t = \langle\langle x\rangle\rangle_t + \langle\langle y\rangle\rangle_t$.

Multiplication Gate: $\langle\langle z\rangle\rangle_t = \mathsf{MGate}(\langle\langle x\rangle\rangle_t, \langle\langle y\rangle\rangle_t, (\langle\langle a\rangle\rangle_t, \langle\langle b\rangle\rangle_t, \langle\langle c\rangle\rangle_t))$

1. Let $\langle\langle x\rangle\rangle_t$ and $\langle\langle y\rangle\rangle_t$ are the inputs to the multiplication gate and $(\langle\langle a\rangle\rangle_t, \langle\langle b\rangle\rangle_t, \langle\langle c\rangle\rangle_t)$ is the random multiplication triple assigned to it.
2. Parties compute $\langle\langle \alpha\rangle\rangle_t = \langle\langle x\rangle\rangle_t - \langle\langle a\rangle\rangle_t$ and $\langle\langle \beta\rangle\rangle_t = \langle\langle y\rangle\rangle_t - \langle\langle b\rangle\rangle_t$.
3. Parties invoke 5VSS-Rec to publicly reconstruct α and β from $\langle\langle \alpha\rangle\rangle_t$ and $\langle\langle \beta\rangle\rangle_t$ respectively.
4. Parties compute $\langle\langle z\rangle\rangle_t = \alpha\beta + \alpha\langle\langle b\rangle\rangle_t + \beta\langle\langle a\rangle\rangle_t + \langle\langle c\rangle\rangle_t$.

Output Gate: $x = \mathsf{OGate}(\langle\langle x\rangle\rangle_t)$

1. Parties invoke 5VSS-Rec to publicly reconstruct x from $\langle\langle x\rangle\rangle_t$ and output x.

Lemma 5.15 *Given $2d^\star$-sharing of $(c_M + c_R)$ secret multiplication triples, each having $\frac{\epsilon}{(2c_M+c_O)}$ error and $2d^\star$-sharing of the inputs of the parties, each having $\frac{\epsilon}{\max(n^2,(2c_M+c_O))}$ error, protocol ComputationPhase correctly evaluates the circuit gate-by-gate in a shared fashion and outputs the desired outputs, except with error probability ϵ.*

Proof: Each multiplication gate requires public reconstruction of two $2d^\star$-sharing, while each output gate requires public reconstruction of one $2d^\star$-sharing. Each such sharing will have at most $\frac{\epsilon}{(2c_M+c_O)}$ error which is maximum of $\frac{\epsilon}{(2c_M+c_O)}$ and $\frac{\epsilon}{\max(n^2,(2c_M+c_O))}$ (this will follow from Linearity of $2d^\star$-sharing). So throughout the computation, reconstruction of secrets from their $2d^\star$-sharing has to be done at most $2c_M + c_O$ times. So in the worst case, the computation of the circuit will generate correct output, except with error probability $\frac{\epsilon}{(2c_M+c_O)}(2c_M + c_O) = \epsilon$.

Moreover, **Preparation Phase** and **Input Phase** will succeed, except with error probability ϵ. Hence **Computation Phase** will have an error probability of ϵ. $\square$

Lemma 5.16 *Protocol* ComputationPhase *achieves the following bounds:*

1. **Round Complexity:** $2\mathcal{D}$ *Rounds, where* $\mathcal{D}$ *is multiplicative depth of the circuit.*

2. **Communication Complexity:** *Private and broadcast communication of* $\mathcal{O}((c_M + c_O)n^3 \log \frac{1}{\epsilon})$ *bits.*

5.6 Statistical MPC Protocol

Now our new statistical MPC protocol for evaluating function f is: (1). Invoke PreparationPhase$(\mathcal{P}, \epsilon)$ and InputPhase$(\mathcal{P}, \epsilon)$ parallely. (2). Invoke ComputationPhase$(\mathcal{P}, \epsilon)$. The protocol works on a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^3 2^{-\kappa} \cdot \max(n^2, (2c_M + c_O))$.

Theorem 5.17 *Except with an error probability* ϵ*, our new statistical MPC protocol can correctly compute an agreed upon function, against an active adversary* $\mathcal{A}_t$ *where* $n = 2t + 1$*. During the protocol, adversary does not get any extra information other than what can be inferred from the input and output of the corrupted parties.*

Theorem 5.18 *Our statistical MPC protocol achieves the following:*

1. **Round Complexity:** $29 + 2\mathcal{D} + 2 = \mathcal{O}(\mathcal{D})$*; 29: For preparation plus input phase,* $2\mathcal{D}$*: For multiplications gates, 2: For output gates.*

2. **Communication Complexity:** *Private and Broadcast communication of* $\mathcal{O}(((c_I + c_R + c_M + c_O)n^3 + n^5) \log \frac{1}{\epsilon})$ *bits.*

3. **Computation Complexity:** $Poly(n, \log \frac{1}{\epsilon}, c_I, c_M, c_A, c_O, c_R)$.

5.7 Conclusion and Open Problems

In this chapter, we presented a new optimally resilient statistical MPC whose round complexity is $\mathcal{O}(\mathcal{D})$ and which broadcasts $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits per multiplication gate. Hence our protocol maintains the round complexity of most round efficient protocol while improving the communication complexity. Moreover, for all functions with constant multiplicative depth, our protocol achieves constant round complexity while most communication efficient MPC of [12] requires $\mathcal{O}(n^2)$ rounds.

The key building blocks of our new MPC are the novel ICP presented in Chapter 2 and a VSS protocol presented in Chapter 4. Using our VSS protocol, we propose a new and *robust* multiplication protocol for generating multiplication triples. We leave the following as an interesting open question:

Open Problem 8 *Can we further improve the communication and round complexity of optimally resilient, statistical MPC protocol in synchronous network?*

Chapter 6

Statistical Multiparty Set Intersection

In information theoretic settings, a protocol for multiparty set intersection (MPSI) allows a set of n parties, each having a set of size m to compute the intersection of those sets, even though t out of the n parties are corrupted by an *active* adversary having *unbounded computing power*. In this chapter, we re-visit the problem of MPSI in information theoretic settings. In information theoretic settings, Li et al. [116] have proposed an statistical MPSI protocol with $n = 3t + 1$ parties. However, we show that the round and communication complexity of the protocol in [116] is much more than what is claimed in [116].

We then propose a new statistical protocol for MPSI with $n = 3t + 1$ parties, which significantly improves the "actual" round and communication complexity of the protocol given in [116]. To design our protocol, we use several tools including a statistical VSS protocol, which are of independent interest.

Both the protocol of [116] and our proposed protocol have non-optimal resilience. So in this chapter, we also present a protocol for statistical MPSI with optimal resilience; i.e., with $n = 2t + 1$. This protocol adapts some of the techniques used in our proposed general statistical MPC protocol presented in Chapter 5. To the best of our knowledge, this is the first ever MPSI protocol with $n = 2t + 1$.

6.1 Introduction

6.1.1 Secure Multiparty Set Intersection (MPSI)

In information theoretic settings, a protocol for multiparty set intersection (MPSI) allows a set of n parties, each having a set of size m to compute the intersection of those sets, even though t out of the n parties are corrupted by an *active* or Byzantine adversary $\mathcal{A}_t$, having *unbounded computing power*. Specifically, let the set of n parties be denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$. Each party P_i has a private data-set S_i, containing m elements from a finite field $\mathbb{F}$. The goal of an MPSI protocol is to compute the intersection of these n sets, satisfying the following properties, even in the presence of $\mathcal{A}_t$:

1. **Correctness:** At the end of the protocol, each honest party correctly gets the intersection of the n sets, irrespective of the behavior of $\mathcal{A}_t$ and

2. **Secrecy:** The protocol should not leak any *extra* information to the corrupted parties, other than what is implied by the input of the corrupted parties (i.e., the data-sets possessed by corrupted parties) and the final output (i.e., the intersection of all the n data-sets).

MPSI problem is a specific instance of MPC problem and also it is an interesting secure distributed computing problem by its own right. It has huge practical applications such as online recommendation services, medical databases, data mining etc. [84, 113].

6.1.2 Existing Literature on MPSI

The MPSI problem was first studied in *cryptographic* model in [84, 113], under the assumption that $\mathcal{A}_t$ has *bounded computing power*. By representing the data-sets as polynomials, the authors of [84, 113] reduced the set intersection problem to the task of securely computing the common roots of n polynomials. The reduction is as follows: Let $S = \{e_1, \ldots, e_m\}$ be a set of size m, where $\forall i, e_i \in \mathbb{F}$. Now set S can be represented by a polynomial $f(x)$ of degree m, where $f(x) = \prod_{i=1}^{m}(x - e_i) = a_0 + a_1x + \ldots + a_mx^m$. It is obvious that if an element e is a root of $f(x)$, then e is a root of $r(x)f(x)$ too, where $r(x)$ is a *random* polynomial of degree m over $\mathbb{F}$. Now for MPSI, party P_i represents his set S_i, by a degree-m polynomial $f^i(x)$ and supplies its $m+1$ coefficients as his input, in a secure manner. Then all the parties jointly and securely compute

$$F(x) = (r^1(x)f^1(x) + r^2(x)f^2(x) + \ldots + r^n(x)f^n(x)) \tag{6.1}$$

where $r^1(x), \ldots r^n(x)$ are n random, secret polynomials of degree m over $\mathbb{F}$, jointly generated by the n parties. Note that $F(x)$ preserves all the common roots of $f^1(x), \ldots, f^n(x)$. Every element $e \in (S_1 \cap S_2 \cap \ldots \cap S_n)$ is a root of $F(x)$, i.e. $F(e) = 0$. Hence after computing $F(x)$ in a secure manner, it can be reconstructed by every party, who locally checks if $F(e) = 0$ for every e in his private set. All the e's at which the evaluation of $F(x)$ is zero form the intersection set $(S_1 \cap S_2 \cap \ldots \cap S_n)$. In [113], it has been proved formally that $F(x)$ does not reveal any *extra* information to the adversary, other than what can be deduced from $(S_1 \cap S_2 \cap \ldots \cap S_n)$ and input set S_i of the corrupted parties. This asserts that the above method of solving MPSI problem perfectly maintains **Secrecy** property (for the sake of completeness, we will prove this later in this chapter). But it is to be noted that the above method satisfies **Correctness** only with very high probability but not perfectly. The reason is that there may exist some $e' \in \mathbb{F}$, such that $F(e') = 0$, even though $e' \notin (S_1 \cap S_2 \cap \ldots \cap S_n)$. These $e' \in \mathbb{F}$ are the roots of $F(x)$ that are not part of the intersection set, but may belong to the private data-sets of some of the honest parties. In [113], it has been proved formally that the roots of $F(x)$ which are not part of intersection set, may belong to the private data-sets of some of the honest parties with negligible probability. For the sake of completeness, we will provide an elaborate proof for this later in this chapter.

In [116], the authors presented the first information theoretically secure protocol for MPSI, assuming $\mathcal{A}_t$ to be *computationally unbounded* and $n = 3t + 1$. Specifically, the authors have shown how to securely compute $F(x)$ in the presence of a computationally unbounded $\mathcal{A}_t$. *Notice that, although not explicitly stated in [116], the MPSI protocol of [116] involves a negligible error probability* *in* **Correctness**. This is due to the argument given above. Hence, the MPSI protocol of [116] is statistical in nature, having a negligible error probability in **Correctness**.

6.1.3 The Network and Adversary Model

An MPSI protocol is executed among a set of n parties, denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$, among which at most t parties can be corrupted by a centralized adversary $\mathcal{A}_t$. In this chapter, we consider two cases: $n = 3t + 1$ as well as $n = 2t + 1$. We assume that each party is directly connected to every other party by a secure channel. The underlying network is assumed to be synchronous. Any protocol in such a network operates in a sequence of rounds. When $n = 3t + 1$, then the availability of physical broadcast channel is optional. If a physical broadcast channel is available in the system, then a broadcast will take one round. Otherwise, we can simulate broadcast using a protocol (for example say protocols of [89, 40]) among the parties in $\mathcal{P}$, which will have the same effect as a physical broadcast channel. The broadcast protocols of [89, 40] requires $\mathcal{O}(t)$ rounds and private communication of $\mathcal{O}(n^2\ell)$ bits to simulate broadcast for ℓ bit message. But while we consider $n = 2t + 1$, we assume the explicit availability of a physical broadcast (recall that it is necessary for any statistical MPC with $n = 2t + 1$).

As in the previous chapters, the adversary that we consider is a *static, threshold, active and rushing* adversary having *unbounded computing power*.

6.1.4 Our Motivation and Contribution

The authors in [116] claimed that their MPSI protocol takes *six* rounds and communicates $\mathcal{O}(n^4m^2)$ elements from $\mathbb{F}$[1]. However, we show that the round and communication complexity of the MPSI protocol of [116] is much more than what is claimed in [116]. We then propose a new, statistical protocol for MPSI with $n = 3t + 1$ parties, which significantly improves the "actual" round and communication complexity of the MPSI protocol given in [116]. The protocol takes *constant number of rounds*, incurs a communication of $\mathcal{O}((m^2n^3 + n^3)\log\frac{1}{\epsilon})$ bits, where each party has a set of size m and the protocol involves an error probability of ϵ. The key tools of our new MPSI are a new statistical VSS and few others special purpose protocols designed with $3t + 1$ parties. Needless to say, our VSS and other tools are of independent interest.

Both the protocol of [116] and our proposed protocol have non-optimal resilience. In fact, in [116], the authors have left it as an open problem to design an MPSI protocol with *optimal resilience*; i.e., with $n = 2t + 1$. So in this chapter, we also present a protocol for statistical MPSI with optimal resilience; i.e., with $n = 2t + 1$ (given a physical broadcast channel). This protocol adapts some of the techniques used in our proposed statistical MPC protocol presented in Chapter 5. The protocol takes *constant number of rounds*, incurs a communication of $\mathcal{O}((m^2n^4 + n^5)\log\frac{1}{\epsilon})$ bits. So our MPSI protocol with optimal resilience requires a communication complexity that is n^2 times more than the communication complexity of the MPSI protocol designed with $n = 3t + 1$. To the best of our knowledge, this is the first ever MPSI protocol with $n = 2t + 1$.

[1]In [116], k is used to denote the size of each set.

6.1.5 The Road-map

In section 6.2, we give a correct estimate of the round and communication complexity of the MPSI protocol of [116]. In section 6.3, we elaborately discuss about our new MPSI protocol with $n = 3t + 1$. In section 6.4, we present a simple protocol for generating random values from field $\mathbb{F}$. In section 6.5, we present our new statistical VSS protocol. In section 6.7, we present a multiplication protocol (an important tool for designing MPSI protocol), along with the required sub-protocol for constructing the multiplication protocol. Subsequently in section 6.8, we present our MPSI protocol with $n = 3t + 1$. Finally in section 6.9, we design our MPSI with optimal resilience. We conclude the chapter with a concluding remark in section 6.10.

6.2 Round and Communication Complexity of MPSI Protocol of [116]

In order to securely compute $F(x)$ given in (6.1) against a computationally unbounded $\mathcal{A}_t$, the MPSI protocol of [116] is divided into three phases: (a) Input Phase, (b) Computation Phase and (c) Output Phase. We briefly recall the steps performed in first two phases (which are the most expensive phases in terms of round and communication complexity) and try to give a correct analysis of those phases.

1. Input Phase: Here each party represents his private data-set S_i as a polynomial say, $f^i(x)$ and t-shares[2] the coefficients of $f^i(x)$ among the n parties. Moreover, each party also t-shares $n(m+1)$ random values which can be assumed as the coefficients of n random polynomials, each of degree m. These $n(m+1)$ random sharings are used to generate the sharings of the coefficients of the secret random polynomials $r^1(x), \ldots, r^n(x)$. To achieve t-sharing, the parties use a VSS protocol. A VSS protocol [43, 137, 91, 73, 109, 125], ensures that a party (possibly corrupted) "consistently" and correctly t-shares a value. So in the **Input Phase** of [116], each party executes $(m+1)$ instances of VSS to share the coefficients of $f^i(x)$. In addition, each party also invokes $n(m+1)$ instances of VSS to generate t-sharing of the coefficients of n random polynomials, each of degree m. So the total number of VSS instances invoked in **Input Phase** is $\mathcal{O}(n^2 m)$.

Now, the authors in [116] claimed that the above steps requires two rounds, where in the first round, each party does the sharing and in the second round verification is done by all parties to ensure whether everybody has received correct and consistent shares (see section 4.2 in [116]). However, no estimation is done for the communication complexity of this phase. Now it is well known that the minimum number of rounds taken by any VSS protocol (that does not involve any error probability) with $n = 3t + 1$ is at least *three* [91, 73, 109]. Moreover, the current best three round VSS protocol with $n = 3t + 1$ requires a private communication and broadcast of $\mathcal{O}(n^3)$ field elements [73, 109]. So far there is no statistical VSS (VSS with negligible error probability) protocol for $n = 3t + 1$ with less than three rounds for generating t-sharing.

[2] We say that an element $c \in \mathbb{F}$ is t-shared among the n parties, if there exists a random polynomial $p(x)$ over $\mathbb{F}$ of degree t such $p(0) = c$ and each (*honest*) party P_i has the share $p(i)$.

Now using the VSS of [73, 109], the **Input Phase** will take at least three rounds, with a private communication and broadcast of $\mathcal{O}(n^5m)$ field elements.

2. Computation Phase: Given that the coefficients of $f^1(x), \ldots, f^n(x), r^1(x), \ldots, r^n(x)$ are t-shared in the **Input Phase**, in the **Computation Phase** the parties jointly try to compute $F(x) = r^1(x)f^1(x) + r^2(x)f^2(x) + \ldots + r^n(x)f^n(x)$, such that the coefficients of $F(x)$ are t-shared. For this, the parties execute a sequence of steps. But we recall only first two steps, which are crucial in the communication and round complexity analysis of the **Computation Phase**.

During **step 1**, the parties locally multiply the shares of the coefficients of $r^i(x)$ and $f^i(x)$, for $i = 1, \ldots, n$. This results in $2t$-sharing[3] of the coefficients of $f^i(x)r^i(x)$ for $i = 1, \ldots, n$. During **step 2**, each party invokes a *re-sharing protocol* and converts the $2t$-sharing of the coefficients of $f^i(x)r^i(x)$ into t-sharing, for $i = 1, \ldots, n$. The re-sharing protocol enables a party to generate t-sharing of an element, given the t'-sharing of the same element, where $t' > t$. In [116], the authors have called a re-sharing protocol, without giving the actual details and claimed that the re-sharing and other additional verifications will take *only three rounds*, with a private communication of $\mathcal{O}(n^4m^2)$ field elements (see section 4.2 of [116]). The authors in [116] have given the reference of [98] for the details of re-sharing protocol. However, the protocol given in [98] is a protocol for general secure MPC, which uses "circuit based approach" to securely evaluate a function. Specifically, the MPC protocol of [98] assumes that the (general) function to be computed is represented as an arithmetic circuit over $\mathbb{F}$, consisting of addition, multiplication, random, input and output gates. The re-sharing protocol of [98] was used to evaluate a multiplication gate. But the protocol was *non-robust* in the sense that it fails to achieve its goal when at least one of the parties misbehaves, in which case the protocol outputs a pair of parties such that at least one of them is corrupted. In fact, the MPC protocol of [98] takes $\Omega(t)$ rounds in the presence of broadcast channel in the system.The authors in [116] have not mentioned what will be the outcome of their protocol if the re-sharing protocol (whose details they have not given) fails during the **Computation Phase**. In fact, computing t-sharing of the coefficients of $F(x)$ by using the ideas of best known general MPC protocol with $n = 3t + 1$ [98, 52, 14] will require a communication complexity of $\Omega(m^2n^2)$ field elements and round complexity of $\Omega(t)$ rounds in the presence of a broadcast channel.

To summarize, a more accurate estimation of the round complexity and communication complexity of the MPSI protocol of [116] in the presence of a physical broadcast channel is as follows:

> In the presence of a physical broadcast channel in the system, the **Input Phase** of the MPSI protocol in [116] will require a private and broadcast communication of $\Omega(n^5m)$ field elements. Moreover, the **Computation Phase** of the MPSI protocol in [116] will take $\Omega(t)$ rounds and communication complexity of $\Omega(m^2n^2)$ field elements.

[3]We say that an element $c \in \mathbb{F}$ is $2t$-shared among the n parties if there exists a polynomial $p(x)$ over $\mathbb{F}$ of degree $2t$, such that $p(0) = c$ and each (*honest*) party P_i has the share $p(i)$.

6.3 Discussion on Our New MPSI Protocol with $n = 3t+1$

We propose a new, information theoretically secure MPSI protocol with $n = 3t+1$, tolerating a *computationally unbounded* $\mathcal{A}_t$. Our protocol is based on the approach of solving the MPSI by securely computing the function given in (6.1). Moreover, our protocol involves a negligible error probability in **Correctness**. However, as mentioned in section 6.1, any protocol for MPSI, based on computing the function in (6.1) will involve a negligible error probability in **Correctness**. In Table 6.1, we compare the round complexity (RC) and communication complexity (CC) of our MPSI protocol with the *estimated* RC and CC of the MPSI protocol of [116] (as stated in previous section).

Table 6.1: Comparison of our MPSI protocol with the MPSI protocol of [116].

Reference	CC in bits		RC
	Private	Broadcast	
[116]	$\Omega((n^5m + m^2n^2)\log(\lvert\mathbb{F}\rvert))$	$\Omega(n^5m\log(\lvert\mathbb{F}\rvert))$	$\Omega(t)$
This Chapter	$\mathcal{O}((m^2n^3 + n^4)\log(\lvert\mathbb{F}\rvert))$	$\mathcal{O}((m^2n^3 + n^4)\log(\lvert\mathbb{F}\rvert))$	37

From the table, we find that our MPSI protocol improves the *estimated* round complexity and communication complexity of the MPSI protocol of [116].

6.3.1 Our MPSI Protocol with $n = 3t+1$ vs. Existing General MPC Protocols

The MPSI problem may be considered as a specific instance of general secure MPC problem [151]. Recall that the generic function f in a MPC is represented as an arithmetic circuit over the finite field $\mathbb{F}$, consisting of five type of gates, namely addition, multiplication, random, input and output. The number of gates of these types are denoted by c_A, c_M, c_R, c_I and c_O respectively. Any general MPC protocol tries to securely evaluate the circuit gate-by-gate, keeping all the inputs and intermediate results of the circuit as t-shared (see [19, 5, 6, 7, 20, 12, 13, 14, 41, 48, 52, 95, 93, 98, 101, 103, 104, 135, 138] and their references).

The MPSI problem can be solved using any general MPC protocol. However, since a general MPC protocol does not exploit the nuances and the special properties of the problem, it is not efficient in general. Moreover, we do not know how to customize the generic MPC protocols to solve MPSI problem in an optimal fashion. However, we outline below a general approach and use the same to estimate the complexity of MPSI protocols, that could have been derived from general MPC protocols.

Assume that an MPSI protocol computes the function given in (6.1), using general MPC protocol. The arithmetic circuit, representing the function in (6.1), will roughly require the following number of gates:

1. $c_I = n(m+1)$ input gates, as every party P_i inputs $(m+1)$ coefficients of $f^i(x)$;
2. $c_R = n(m+1)$ random gates, as n polynomials $r^1(x), \ldots, r^n(x)$ will have $n(m+1)$ random coefficients in total;
3. $c_M = n(m+1)^2$ multiplication gates. This is because computing $r^i(x)f^i(x)$ requires $(m+1)^2$ coefficient multiplications;

4. $c_O = 2m + 1$ output gates, as $2m + 1$ coefficients of $F(x)$ should be output.

In Table 6.2, we give the round complexity (RC) and communication complexity (CC) of best known general MPC protocols with $n = 3t + 1$, to securely compute the function (6.1), represented by above number of gates.

Table 6.2: Comparison of our MPSI with the general MPC protocols that securely compute (6.1).

Reference	CC in bits		RC
	Private	Broadcast	
[20]	$\mathcal{O}(n^5 m^2 \log(\|\mathbb{F}\|))$	$\mathcal{O}(n^5 m^2 \log(\|\mathbb{F}\|))$	$\mathcal{O}(1)$
[98]	$\mathcal{O}(n^4 m^2 \log(\|\mathbb{F}\|))$	$\mathcal{O}(n^2 \log(\|\mathbb{F}\|))$	$\mathcal{O}(n)$
[52]	$\mathcal{O}(n^2 m^2 \log(\|\mathbb{F}\|))$	$\mathcal{O}(n^2 \log(\|\mathbb{F}\|))$	$\mathcal{O}(n)$
[14]	$\mathcal{O}(n^2 m^2 \log(\|\mathbb{F}\|))$	$\mathcal{O}(n^3 \log(\|\mathbb{F}\|))$	$\mathcal{O}(n)$
This chapter	$\mathcal{O}((m^2 n^3 + n^4) \log(\|\mathbb{F}\|))$	$\mathcal{O}((m^2 n^3 + n^4) \log(\|\mathbb{F}\|))$	37

From Table 6.2, we find that our protocol incurs much lesser communication complexity than the protocol of [20], while keeping the round complexity same. But the protocols of [98, 52, 14] provide slightly better communication complexity than ours at the cost of increased round complexity. Round complexity and communication complexity are two important parameters of any distributed protocol. Therefore, if we ever hope to practically implement MPSI protocols, then we should look for a solution that tries to *simultaneously minimize* both these parameters.

Though our main motive in this chapter is to present a clean solution for MPSI, as a bi-product we have shown that our protocol simultaneously improves both communication and round complexity, whereas existing general MPC protocols (when applied to solve MPSI) improve only one of these two parameters.

6.3.2 The Working Field of our MPSI Protocol

Our statistical MPSI protocol involves a negligible error probability of ϵ in correctness property. To bound the error probability by ϵ, all the computations in our protocol are performed over a finite field $\mathbb{F} = GF(2^\kappa)$,where κ has to be determined using the relation $\epsilon \geq \max(m^2, n)n^3 2^{-\kappa}$. We assume that $n = \text{poly}(m)$. Any field element from field $\mathbb{F}$ can be represented by κ bits, where $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ (this can be derived using $n = \mathcal{O}(\log \frac{1}{\epsilon})$ and $n = \text{poly}(m)$).

In order to bound the error probability of our MPSI protocol by some specific value of ϵ, we find out the *minimum* value of κ that satisfies $\epsilon \geq \max(m^2, n)n^3 2^{-\kappa}$. This value for κ will consequently determine the field $\mathbb{F}$ over which our protocol should work.

6.3.3 Overview of Our Protocol

As mentioned earlier, our MPSI protocol tries to securely compute the function given in (6.1). Our protocol is divided into four phases, namely (a) Input Phase; (b) Preparation Phase; (c) Computation Phase and (d) Output Phase. In the Input phase, the parties t-share the coefficients of their input polynomials. In the

Preparation phase, the parties jointly generate the t-sharing of the secret random $r^i(x)$ polynomials. To achieve the task in Input phase, we design a new statistical VSS protocol, called VSS that uses MVMS-ICP presented in Chapter 2. Note that we can not use our 2-round sharing 2-round reconstruction $(3t+1, t)$ statistical VSS protocol, namely protocol 2-Round-VSS, presented in section 3.4 of Chapter 3 for our purpose here. This is because as mentioned in Chapter 3, protocol 2-Round-VSS follows weak definition of statistical VSS (see Definition 3.3) and also it does not generate t-sharing of secret. Now the task in Preparation phase is achieved by a sub-protocol called Random that uses VSS as building block.

In the Computation Phase, the parties generate the t-sharing of the coefficients of $r^i(x)f^i(x)$. For this, we use sub-protocol Mult, which is a combination of few existing ideas from the literature and few new ideas presented in this chapter. Finally, in the Output Phase, the coefficients of $F(x)$ are reconstructed by each party.

Most of the sub-protocols presented in this chapter, are designed to *concurrently* deal with $\ell \geq 1$ values. We can show that our sub-protocols, concurrently dealing with ℓ values, are better in terms of communication complexity, than ℓ concurrent executions of the existing sub-protocols working with single value. Thus, our sub-protocols harness the advantage offered by dealing with multiple values concurrently.

6.4 Generation of a Random Value

We now present a protocol called RandomVector($\mathcal{P}$), which allows the parties in $\mathcal{P}$ to jointly generate a random element from $\mathbb{F}$. Protocol RandomVector uses the four round perfect VSS protocol of [91] (see Fig 2 of [91]) as black box. The perfect VSS with $n = 3t+1$ parties consists of two phases, namely Sharing Phase and Reconstruction Phase. The Sharing Phase takes four rounds and allows a dealer D (which can be any party from the set of n parties) to verifiably share a secret $s \in \mathbb{F}$ by privately communicating $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits and broadcasting $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits where $|\mathbb{F}| \geq n$. The Reconstruction Phase takes single round and allows all the (honest) parties to reconstruct the secret s (shared by D in Sharing Phase) by broadcasting $\mathcal{O}(n \log |\mathbb{F}|)$ bits in total. Notice that, in our context, $|\mathbb{F}| = 2^\kappa \geq n$. The protocol is given in Fig. 6.1.

Lemma 6.1 *Protocol RandomVector generates a random value in five rounds. The protocol privately communicates and broadcasts $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits.*

PROOF: Communication and round complexity is easy to see. The correctness follows from the correctness of the four round perfect VSS of [91] and the above discussion. □

6.5 Statistical VSS with $n = 3t + 1$

In this section, we present a new statistical VSS protocol with $n = 3t+1$ parties that can share/commit ℓ secrets concurrently. It follows the strong definition of VSS (see Definition 3.2) as opposed to protocol 2-Round-VSS of Chapter 3 that follows only the weak definition of VSS (see Definition 3.1). Hence we can not use protocol 2-Round-VSS for our purpose here. Our VSS protocol presented in this section has an error probability of ϵ.

Figure 6.1: Protocol RandomVector: Generates a random value.

RandomVector($\mathcal{P}$)

1. Every party $P_i \in \mathcal{P}$ selects a random element r^i from $\mathbb{F}$.
2. Every party $P_i \in \mathcal{P}$ as a dealer invokes Sharing Phase of four round VSS protocol of [91] with $n = 3t + 1$ for sharing r^i.
3. For reconstructing the values r^i (shared by P_i in Sharing Phase), the Reconstruction Phase of four round VSS of [91] with $n = 3t + 1$ is invoked. Now corresponding to every $P_i \in \mathcal{P}$, the values r^i are public.
4. Now parties compute $r = \sum_{i=1}^{n} r^i$. It is clear that r will be a random value from $\mathbb{F}$.

To bound the error probability by ϵ, the computation in our statistical VSS protocol is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq \max(\ell, n^2)2^{-\kappa}$. In our VSS protocol, MVMS-ICP will be invoked with $\frac{\epsilon}{n}$ error probability and as mentioned in previous section $\epsilon \geq n2^{-\kappa}$ should hold to bound error probability of MVMS-ICP by ϵ. This implies that $\epsilon \geq n^2 2^{-\kappa}$. During the discussion of our protocol, we will show that ϵ should also satisfy $\epsilon \geq \ell 2^{-\kappa}$ to bound the error probability of our VSS protocol by ϵ. Combining both the relations we get, $\epsilon \geq \max(\ell, n^2)2^{-\kappa}$. So here each element from the field is represented by $\kappa = \log|\mathbb{F}| = \mathcal{O}(\log \frac{\max(\ell,n^2)}{\epsilon}) = \mathcal{O}(\log n + \log \frac{1}{\epsilon}) = \mathcal{O}(\log \frac{1}{\epsilon})$ bits (this follows from $n = \mathcal{O}(\log \frac{1}{\epsilon})$ and our assumption that $\ell = poly(n)$).

The Intuition: Informally, our VSS protocol called as VSS works as follows: In the sharing phase, D on having ℓ secrets $(s^1, \ldots, s^\ell)$ chooses $\ell + 1$ random polynomials $f^0(x), \ldots, f^\ell(x)$ over $\mathbb{F}$, each of degree t, such that $f^0(0) = s^0$ and $f^l(0) = s^l$ for $l = 1, \ldots, \ell$. Here s^0 is a random non-zero element from $\mathbb{F}$. D then hands over his IC signature on i^{th} points of $\ell+1$ polynomials *concurrently* to party P_i. After this, the parties jointly produce a non-zero random value z. Now D is asked to broadcast a linear combination of the $\ell + 1$ polynomials. Specifically, D broadcasts $f(x) = \sum_{l=0}^{\ell} f^l(x)z^l$. Now each party P_i has i^{th} value on each of the $f^l(x)$ polynomials. Thus with those values P_i can compute $y_i = \sum_{l=0}^{\ell} f^l(i)z^l$ and check whether indeed $y_i = f(i)$ holds or not. If the condition is not satisfied the P_i reveals the IC signature received from D on the i^{th} values of the polynomials $f^l(x)$. If P_i is successful in revealing the IC signature and indeed $y_i \neq f(i)$, then D is discarded (and therefore ℓ predefined values are taken as D's secret). Otherwise, everybody assumes that D has correctly committed ℓ secrets, with very high probability. The protocol for sharing phase is formally given in Fig. 6.2.

Reconstruction phase of VSS (presented in Fig. 6.3) can be easily implemented using Reed-Solomon Error correction algorithm (e.g. Berlekamp Welch Algorithm [119]).

We now prove the properties of our VSS scheme.

Claim 6.2 *An honest D will not be* **discarded** *in sharing phase protocol VSS-*

Figure 6.2: Protocol VSS-Share: Sharing Phase of Protocol VSS.

VSS-Share$(D, \mathcal{P}, \ell, (s^1, \ldots, s^\ell), \epsilon)$

1. D chooses a random, non-zero element s^0 from $\mathbb{F}$. Now for $l = 0, \ldots, \ell$, D picks a random polynomial $f^l(x)$ over $\mathbb{F}$ of degree t, with $f^l(0) = s^l$. For $i = 1, \ldots, n$, let $S_i = (s_i^0, s_i^1, \ldots, s_i^\ell)$, where $s_i^l = f^l(i)$ for $l = 0, \ldots, \ell$.
   (a) D sends $ICSig(D, P_i, \mathcal{P}, S_i)$ having $\frac{\epsilon}{n}$ error to party P_i.
   (b) Every party P_i receives $ICSig(D, P_i, \mathcal{P}, S_i)$ having $\frac{\epsilon}{n}$ error from D.
2. All the parties in $\mathcal{P}$ invoke RandomVector($\mathcal{P}$) to generate a random value $z \in \mathbb{F}$.
3. D broadcasts the polynomial $f(x) = \sum_{l=0}^{\ell} f^l(x) z^l$. If the polynomial $f(x)$ broadcasted by D is of degree more than t, then D is discarded and the protocol terminates here.
4. Every party P_i computes $y_i = \sum_{l=0}^{\ell} s_i^l z^l$ and checks whether $f(i) \stackrel{?}{=} y_i$. If no, then P_i reveals $ICSig(D, P_i, \mathcal{P}, S_i)$ having $\frac{\epsilon}{n}$ error. If P_i succeeds to produce the IC signature and $f(i) \neq \sum_{l=0}^{\ell} s_i^l z^l$, then D is discarded and the protocol terminates here.

Figure 6.3: Protocol VSS-Rec: Reconstruction Phase of Protocol VSS

VSS-Rec$(\mathcal{P}, (s^1, \ldots, s^\ell), \epsilon)$

Each party P_i broadcasts his share s_i^l of s^l for all $l = 1, \ldots, \ell$. The parties apply error correction algorithm (e.g. Berlekamp Welch Algorithm [119]) to reconstruct s^l from the n shares.

Share, with probability at least $(1 - \epsilon)$.

PROOF: If D is honest, then he will never broadcast a polynomial $f(x)$ of degree more than t. Now it is clear that an honest D will be discarded if somehow any corrupted party P_i (there are at most t such parties) is able to reveal $ICSig(D, P_i, \mathcal{P}, \overline{S_i})$ such that $\overline{S_i} = (\overline{s_i^0}, \ldots, \overline{s_i^\ell})$ and $f(i) \neq \sum_{l=0}^{\ell} \overline{s_i^l} z^l$. We show that this can happen only with probability at most ϵ.

By **ICP-Correctness3**, a corrupted P_i will be successful in revealing $ICSig$ $(D, P_i, \mathcal{P}, \overline{S_i})$ with $\overline{S_i} \neq S_i$, with probability $\frac{\epsilon}{n}$ (recall that each IC signature has $\frac{\epsilon}{n}$ error). As there are t corrupted parties, the event that some corrupted party will be able to reveal $ICSig(D, P_i, \mathcal{P}, \overline{S_i})$ with $\overline{S_i} \neq S_i$ may occur with probability at most $t\frac{\epsilon}{n} \approx \epsilon$. Hence the claim. □

Claim 6.3 *If D is not discarded in VSS-Share, then there exists $\ell + 1$ unique polynomials $f^0(x), \ldots, f^\ell(x)$ each of degree t, such that for all $l = 1, \ldots, \ell$, s_i^l held by every honest P_i at the end of sharing phase satisfies $f^l(i) = s_i^l$ with probability at least $(1 - \epsilon)$.*

PROOF: Assuming that D is not discarded in VSS-Share, the above lemma holds when D is honest without any error probability. We now consider the case, when D is corrupted and for at least one l, the s_i^l values held by the honest parties lie on a polynomial of degree higher than t. Let H be the set of honest parties in $\mathcal{P}$. Moreover, let $h^0(x), \ldots, h^\ell(x)$ denote the minimum degree polynomial, defined by the points on $f^0(x), \ldots, f^\ell(x)$ respectively, held by the parties in H. Then according to our assumption, degree of at least one of the polynomials $h^0(x), \ldots, h^\ell(x)$ is more than t. Moreover, notice that the degree of $h^0(x), \ldots, h^\ell(x)$ can be at most $|H| - 1$. This is because $|H|$ distinct points can define a polynomial of degree at most $|H| - 1$. Now the value y_i of an honest P_i can be defined as $y_i = \sum_{j=0}^{\ell} z^j h^j(i)$. Let $h^{min}(x)$ be the minimum degree polynomial defined by y_i's, corresponding to $P_i \in H$.

We next claim that if degree of at least one of $h^0(x), \ldots, h^\ell(x)$ is more than t, then $h^{min}(x)$ will be of degree more than t, with very high probability. This will clearly imply that $f(x) \neq h^{min}(x)$ (as $f(x)$ is a polynomial of degree t) and hence $y_i \neq f(i)$, for at least one $P_i \in H$ which is a contradiction, as other wise honest P_i would have revealed $ICSig(P_i, D, \mathcal{P}, S_i)$, except with error probability $\frac{\epsilon}{n}$ and D would have been discarded.

So we proceed to prove that $h^{min}(x)$ will be of degree more than t, when one of $h^0(x), \ldots, h^\ell(x)$ has degree more than t. For this, we show the following:

1. We first show that $h^{def}(x) = \Sigma_{j=0}^{\ell} z^j h^j(x)$ will of degree more than t with probability at least $(1 - \epsilon)$, if one of $h^0(x), \ldots, h^\ell(x)$ has degree more than t.

2. We then show that $h^{min}(x) = h^{def}(x)$, implying that $h^{min}(x)$ will be of degree more than t.

So we proceed to prove the first point. Assume that m is such that $h^m(x)$ has maximal degree among $h^0(x), \ldots, h^\ell(x)$, and let t_m be the degree of $h^m(x)$. Then according to our assumption, $t_m > t$. Also recall that $t_m < |H|$. This is because given $|H|$ values (recall that $h^0(x), \ldots, h^\ell(x)$ are defined by the points on polynomials $f^0(x), \ldots, f^\ell(x)$, held by the honest parties in H), the maximum degree polynomial that can be defined using them is $|H| - 1$. Now each $h^i(x)$ can be written as $h^i(x) = c_{t_m}^i x^{t_m} + \widehat{h^i}(x)$ where $\widehat{h^i}(x)$ has degree lower than t_m. Thus $h^{def}(x)$ can be written as:

$$
\begin{aligned}
h^{def}(x) &= [c_{t_m}^0 x^{t_m} + \widehat{h^0}(x)] + z[c_{t_m}^1 x^{t_m} + \widehat{h^1}(x)] + \ldots + z^\ell [c_{t_m}^\ell x^{t_m} + \widehat{h^\ell}(x)] \\
&= x^{t_m}(c_{t_m}^0 + \ldots + z^\ell c_{t_m}^\ell) + \Sigma_{j=0}^{\ell} z^j \widehat{h^j}(x) \\
&= x^{t_m} c_{t_m} + \Sigma_{j=0}^{n} z^j \widehat{h^j}(x)
\end{aligned}
$$

By assumption $c_{t_m}^m \neq 0$. It implies that $(c_{t_m}^0, \ldots, c_{t_m}^\ell)$ is not a complete 0 vector. Hence $c_{t_m} = c_{t_m}^0 + \ldots + z^\ell c_{t_m}^\ell$ will be zero with probability $\frac{\ell}{|\mathbb{F}|-1} \leq \frac{\ell\epsilon}{\max(\ell, n^2)} \leq \epsilon$ (to bound $\frac{\ell}{|\mathbb{F}|-1}$ by ϵ, we require $\epsilon \geq \ell 2^{-\kappa}$; that is why we had set $\epsilon \geq \max(\ell, n^2)2^{-\kappa}$). This is because $(c_{t_m}^0, \ldots, c_{t_m}^\ell)$ may be considered as the set of coefficients of a degree-ℓ polynomial, say $\mu(x)$, and hence the value c_{t_m} is the value of $\mu(x)$ evaluated at $x = z$. Now c_{t_m} will be zero if z happens to be one of the ℓ roots of $\mu(x)$ (since degree of $\mu(x)$ is at most ℓ). Since z is generated randomly from $\mathbb{F} \setminus \{0\}$, independent of $h^0(y), \ldots, h^\ell(x)$, the probability that it is a root of $\mu(x)$ is $\frac{\ell}{|\mathbb{F}|-1} \leq \epsilon$. So with probability at least $(1-\epsilon)$, c_{t_m}, which is the t_m^{th} coefficient of $h^{def}(x)$ is non-zero. This implies that $h^{def}(x)$ will be of degree $t_m > t$ with probability $(1-\epsilon)$. Notice that each y_i of an honest P_i, will lie on $h^{def}(x)$.

Now we will show that $h^{min}(x) = h^{def}(x)$ and thus $h^{min}(x)$ has degree at least t_m, which is greater than t. So consider the difference polynomial $dp(x) = h^{def}(x) - h^{min}(x)$. Clearly, $dp(x) = 0$, for all $x = i$, where $P_i \in H$. Thus $dp(x)$ will have at least $|H|$ roots. On the other hand, maximum degree of $dp(x)$ could be t_m, which is at most $|H|-1$. These two facts together imply that $dp(x)$ is the zero polynomial, implying that $h^{def}(x) = h^{min}(x)$ and thus $h^{min}(x)$ has degree $t_m > t$. □

Remark 6.4 (D's Commitment in VSS-Share) *The polynomials $f^1(x), \ldots, f^\ell(x)$ defined in Claim 6.3 are called D's committed polynomials in protocol VSS-Share. The values $(s^1, \ldots, s^\ell)$ with $s^l = f^l(0)$ are called D's commitment in VSS-Share.*

Claim 6.5 *In protocol VSS-Rec, polynomial $f_i^l(x)$ for all $l = 1, \ldots, \ell$ will be reconstructed with probability at least $(1-\epsilon)$, where $f^1(x), \ldots, f^\ell(x)$ are D's committed polynomials in VSS-Share.*

Proof: From Claim 6.3, D's committed polynomials are of degree t with probability at least $(1-\epsilon)$. Now by the property of Error correction [119, 121], $3t+1$ points on a degree-t polynomial, out of which at most t could be corrupted are enough to correctly reconstruct the polynomial. Hence the polynomials will be reconstructed correctly with probability at least $(1-\epsilon)$ as $n = 3t+1$. □

Lemma 6.6 (Secrecy) *Protocol VSS-Share satisfies perfect secrecy.*

Proof: Here we have to consider the case when D is honest. The adversary $\mathcal{A}_t$ will know only t shares for each $s^i, 0 \leq i \leq n$ from t corrupted parties under its control. Now, $f(0) = \sum_{l=0}^{\ell} s^l z^l$. This implies that the linear combination of the secrets i.e $\sum_{l=1}^{\ell} s^l z^l$ is blinded with a random value s^0, chosen by honest D. Thus, $f(0)$ will look completely random for $\mathcal{A}_t$. This shows that $s^1, \ldots, s^\ell$ will remain information theoretically secure from $\mathcal{A}_t$.

Lemma 6.7 (Correctness) *Protocol VSS satisfies correctness property with probability at least $(1-\epsilon)$.*

Proof: Here we have to consider the case when D is honest. By Claim 6.2, honest D will never be discarded in sharing phase, except with probability ϵ. Now by Claim 6.3, D will commit polynomials $f^1(x), \ldots, f^\ell(x)$ and by Claim 6.5 for all $l = 1, \ldots, \ell$, $f^l(x)$ will be reconstructed with probability at least $(1-\epsilon)$. Hence $s^l = f^l(0)$ for all l will be reconstructed with probability at least $(1-\epsilon)$. □

Lemma 6.8 (Strong Commitment) *Protocol VSS satisfies strong commitment property with probability at least $(1-\epsilon)$.*

PROOF: Here we have to consider the case when D is corrupted. If D is **discarded** during sharing phase then strong commitment holds trivially, as every party may assume ℓ predefined default values as D's commitment. On the other hand, when D is not discarded, the proof follows from the same argument as given in Lemma 6.7. □

Theorem 6.9 *Protocol VSS is an efficient $(3t+1, t)$ statistical VSS protocol.*

PROOF: This follows from Lemma 6.6, 6.7 and 6.8. □

Theorem 6.10 *In protocol VSS, the sharing phase protocol VSS-Share requires eight rounds and the reconstruction phase protocol VSS-Rec requires one round.*

PROOF: We first analyze the round complexity of VSS-Share. Step 1 and 2 of VSS-Share can be executed in parallel. Step 1 requires three rounds (calls several instances of Gen followed by several instances of Ver) and step 2 requires five rounds (invokes one instance of RandomVector). Since step 1 and 2 are executed in parallel, they will require five rounds. Step 3 requires one round and step 4 requires two rounds (may invoke Reveal). Hence in total VSS-Share requires eight rounds.

It is easy to see that protocol VSS-Rec requires one round. □

Theorem 6.11 *Protocol VSS achieves following communication complexity bounds:*

- *VSS-Share requires both private as well as broadcast communication of $\mathcal{O}((\ell n + n^3)\log\frac{1}{\epsilon})$ bits.*
- *Protocol VSS-Rec requires broadcast communication of $\mathcal{O}(\ell n \log\frac{1}{\epsilon})$ bits.*

PROOF: The communication complexity of VSS-Share follows from the fact that there are at most $\mathcal{O}(n)$ executions of Gen and Ver. Moreover VSS-Share invokes one instance of RandomVector. The communication complexity of VSS-Rec follows from the fact that every party broadcasts ℓ elements from $\mathbb{F}$. □

6.5.1 The Output Generated by VSS-Share

At a glance the situation created at the end of VSS-Share is as follows (if D is not **discarded**): There are ℓ degree t polynomials $f^1(x), \ldots, f^\ell(x)$ such that every *honest* party P_i holds values $s_i^l = f^l(i)$, for $l = 1, \ldots, \ell$. For the ease of reference, we use the following definition to capture the output of VSS-Share:

Definition 6.12 (t-$(1d)$-sharing) *We say that a secret $s \in \mathbb{F}$ is t-$(1d)$-shared (here $1d$ stands for one-dimensional) among the parties in $\mathcal{P}$, if the following holds:*

1. *There exists degree t polynomial $f(x)$ with $f(0) = s$;*
2. *The i^{th} value on $f(x)$, namely $s_i = f(i)$, also called as i^{th} share of s, is held by party $P_i \in \mathcal{P}$.*

We denote this by $[s]_t$. If a specific party P does the sharing then we denote it by $[s]_t^P$.

It is easy to see that D has t-$(1d)$-shared ℓ secrets $s^1, \ldots, s^\ell$ at the end of VSS-Share. Moreover, given a t-$(1d)$-sharing of a secret, it can be reconstructed using VSS-Rec (though the protocol has been designed to handle ℓ secrets, it can be easily modified to handle one secret).

Notice that t-$(1d)$-sharing of each s^i (separately) can be produced using a perfect (i.e., without any error) VSS protocol with $n = 3t+1$ [91, 73, 109]. However, this will involve more communication complexity (at least $\Omega(\ell n^2)$) than VSS-Share which performs the same task with less communication complexity (but with a negligible error probability).

Notation 6.13 *We now define few notations which are used in subsequent sections (these notations are also commonly used in the literature of MPC). By saying that parties in $\mathcal{P}$ compute (locally) $([y^1]_t, \ldots, [y^{\ell'}]_t) = \varphi([x^1]_t, \ldots, [x^\ell]_t)$ (for any function $\varphi : \mathbb{F}^\ell \rightarrow \mathbb{F}^{\ell'}$), we mean that each P_i computes $(y_i^1, \ldots, y_i^{\ell'}) = \varphi(x_i^1, \ldots, x_i^\ell)$, where x_i^l and y_i^l denote the i^{th} shares of x^l and y^l respectively. Note that applying an affine (linear) function φ to a number of t-$(1d)$-sharings, we get t-$(1d)$-sharings of the outputs. So by adding two t-$(1d)$-sharings of secrets, we get t-$(1d)$-sharing of the sum of the secrets, i.e. $[a]_t + [b]_t = [a+b]_t$. However, by multiplying two t-$(1d)$-sharings of secrets, we get $2t$-$(1d)$-sharing of the product of the secrets, i.e. $[a]_t[b]_t = [ab]_{2t}$.*

Now in the next section, we will present a simple protocol for generating t-$(1d)$-sharing of a number of random secrets. The protocol uses VSS-Share as a black box.

6.6 Generating Random t-$(1d)$-sharing

We now present a protocol called Random, which allows the parties in $\mathcal{P}$ to jointly generate t-$(1d)$-sharing of ℓ random secrets, i.e $[r^1]_t, \ldots, [r^\ell]_t$, where each r^i is a random element from $\mathbb{F}$. Moreover, the adversary will have no information about the random elements.

Protocol Random is very simple. Here every party as a dealer initiates one instance of VSS-Share to t-$(1d)$-share ℓ secrets, say $(r^{1i}, \ldots, r^{\ell i})$. Let $Pass$ be the set of parties who are not discarded during their corresponding instances of VSS-Share. Then every party locally computes $[r^l]_t = \sum_{P_i \in Pass} [r^{li}]_t$ for all $l = 1, \ldots, \ell$.

To bound the error probability by ϵ, the computation in protocol Random is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq \max(\ell, n^2)n2^{-\kappa}$. This is because Random invokes VSS-Share with $\frac{\epsilon}{n}$ error probability and as mentioned in previous section, $\epsilon \geq \max(\ell, n^2)2^{-\kappa}$ should hold to bound error probability of VSS by ϵ. Each element from the field is represented by $\kappa = \log|\mathbb{F}| = \mathcal{O}(\log\frac{1}{\epsilon})$ bits (this follows from $n = \mathcal{O}(\log\frac{1}{\epsilon})$ and our assumption that $\ell = poly(n)$). Now the protocol is given in Fig. 6.4.

Lemma 6.14 *Protocol Random satisfies the following properties:*

1. **Correctness**: *Except with probability ϵ, Random outputs correct t-$(1d)$-sharing of ℓ random values.*

2. **Secrecy**: *The ℓ values whose t-$(1d)$-sharing is generated by the protocol will be completely random and unknown to $\mathcal{A}_t$.*

Figure 6.4: Protocol Random: Generates t-$(1d)$-sharing of ℓ random secrets.

Random$(\mathcal{P}, \ell, \epsilon)$

1. Every party $P_i \in \mathcal{P}$ invokes VSS-Share$(P_i, \mathcal{P}, \ell, (r^{1i}, \ldots, r^{\ell i}), \frac{\epsilon}{n})$ to t-$(1d)$-share ℓ random elements $r^{1i}, \ldots, r^{\ell i}$ from $\mathbb{F}$.
2. Let $Pass$ be the set of parties who are not discarded in their instance of VSS-Share.
3. If $|Pass| \geq 2t+1$, all the parties in $\mathcal{P}$ jointly compute $[r^l]_t = \sum_{P_i \in Pass} [r^{li}]_t$ for $l = 1, \ldots, \ell$.

PROOF: **Correctness:** An honest P_i will be able to produce $[r^{1i}]_t^{P_i}, \ldots, [r^{\ell i}]_t^{P_i}$, except with error probability $\frac{\epsilon}{n}$. This is because with probability at most $\frac{\epsilon}{n}$, honest P_i might get discarded during VSS-Share (see Claim 6.2) in which case P_i will not be included in $Pass$. Since there are $2t+1$ honest parties, none of them will figure in $Pass$ with probability $(2t+1)\frac{\epsilon}{n} \approx \epsilon$. This will result in $|Pass| \leq t$ and hence output will not be computed, with probability at most ϵ. Hence, except with probability ϵ, Random will generate its desired output.

Secrecy: From the **Secrecy** property of VSS-Share, the values which are t-$(1d)$-shared by an honest party are completely random and are unknown to $\mathcal{A}_t$. $Pass$ will definitely contain at least $t+1$ honest party. Now since addition preserves randomness, $r^1, \ldots, r^\ell$ will be completely random and unknown to $\mathcal{A}_t$. This proves **Secrecy** property. □

Lemma 6.15 *Protocol Random has the following bounds:*

1. **Round Complexity:** *Eight Rounds.*
2. **Communication Complexity:** *Private and broadcast communication of* $\mathcal{O}((\ell n^2 + n^4)\log\frac{1}{\epsilon})$ *bits.*

PROOF: There are n parallel invocations of VSS-Share and each invocation requires eight rounds (see Theorem 6.10) and communication of $\mathcal{O}((\ell n + n^3)\log\frac{1}{\epsilon})$ bits, both private as well as broadcast (see Theorem 6.11). Hence the lemma. □

6.7 Multiplication Protocol

In this section, we present a multiplication protocol which allows the parties to generate $[a^1]_t, \ldots, [a^\ell]_t, [b^1]_t, \ldots, [b^\ell]_t$ and $[c^1]_t, \ldots, [c^\ell]_t$, where a^l's and b^l's are random and $c^l = a^l b^l$ for $l = 1, \ldots, \ell$. Before presenting our multiplication protocol, we present two important protocols that will be used as building block for our multiplication protocol.

6.7.1 Upgrading t-$(1d)$-sharing to t-$(2d)$-sharing

We start with the definition of t-$(2d)$-sharing of secret(s)[4]:

[4]2d stands for two dimensional.

Definition 6.16 (**t-($2d$)-sharing**) *We say that a value s is t-($2d$)-shared (here $2d$ stands for two dimensional) among the parties in $\mathcal{P}$, if the following hold:*

1. *There exists degree t polynomials $f(x), f^1(x), \ldots, f^n(x)$ with $f(0) = s$ and for $i = 1, \ldots, n$, $f^i(0) = f(i) = s_i$.*

2. **Every honest party** *$P_i \in \mathcal{P}$ holds a share $s_i = f(i)$ of s, the polynomial $f^i(x)$ and share-share $s_{ji} = f^j(i)$ for the share s_j of every other (honest) party P_j. In other words, s and every s_i such that P_i is honest is t-($1d$)-shared among the parties in $\mathcal{P}$.*

We denote t-($2d$)-sharing of secret s by $[[s]]_t$.

We now present a new protocol, called Upgrade1dto2d for upgrading t-($1d$)-sharing to t-($2d$)-sharing. That is, given t-($1d$)-sharing of ℓ secrets, namely $[s^1]_t, \ldots, [s^\ell]_t$, Upgrade1dto2d outputs t-($2d$)-sharing $[[s^1]]_t, \ldots, [[s^\ell]]_t$, except with probability of $(1 - \epsilon)$. Moreover, $\mathcal{A}_t$ learns nothing about the secrets during Upgrade1dto2d.

To bound the error probability by ϵ, the computation in protocol Upgrade1dto2d is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq \max(\ell, n^2)n2^{-\kappa}$. This is because Upgrade1dto2d invokes Random with ϵ error probability and VSS-Share with $\frac{\epsilon}{n}$ error probability. Both of them enforces that $\epsilon \geq \max(\ell, n^2)n2^{-\kappa}$ should hold. Each element from the field is represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits. Now the protocol is given in Fig. 6.5.

Lemma 6.17 *Protocol Upgrade1dto2d satisfies the following properties:*

1. **Correctness***: Except with probability ϵ, Upgrade1dto2d outputs correct t-($2d$)-sharing of ℓ values, given their t-($1d$)-sharings.*

2. **Secrecy***: The ℓ values whose t-($2d$)-sharing is generated in protocol Upgrade1dto2d, will remain unknown to $\mathcal{A}_t$.*

PROOF: **Correctness:** First of all, in protocol Upgrade1dto2d, Random will work correctly, except with probability ϵ. Now every honest party P_i will t-($1d$)-share the values $(s_i^0, s_i^1, \ldots, s_i^\ell)$ without being discarded, except with error probability $\frac{\epsilon}{n}$. Since there are $2t + 1$ honest parties, all of them will generate t-($1d$)-sharing of their values without being discarded, except with probability $(2t + 1)\frac{\epsilon}{n} \approx \epsilon$.

Now we show that if a corrupted party P_c t-($1d$)-shares values $\overline{s_c^0}, \overline{s_c^1}, \ldots, \overline{s_c^\ell}$ with $\overline{s_c^l} \neq s_c^l$ for some $l \in \{0, 1, \ldots, \ell\}$, then P_c will be detected to be corrupted with probability at least $(1 - \frac{\epsilon}{n})$. For every honest party P_h, the value $s_h = s_h^0 + \sum_{l=1}^{\ell} r^l s_h^l$ will be reconstructed correctly, where s_h is the h^{th} share of $s = s^0 + \sum_{l=1}^{\ell} r^l s^l$. But for corrupted party P_c, the probability that $\overline{s_c} = \overline{s_c^0} + \sum_{l=1}^{\ell} r^l \overline{s_c^l}$ will be equal to s_c (which is the actual c^{th} share of s) is only $\frac{\ell}{|\mathbb{F}-1|}$. This is same as the probability that two polynomials of degree ℓ with coefficients as $(\overline{s_c^0}, \ldots, \overline{s_c^\ell})$ and $(s_c^0, \ldots, s_c^\ell)$ have same value at random r. Notice that here r has to be generated only after P_c generates the t-($1d$)-sharing of $(\overline{s_c^0}, \ldots, \overline{s_c^\ell})$. Now since $\epsilon \geq \max(\ell, n^2)n2^{-\kappa}$, we have $\frac{\ell}{|\mathbb{F}-1|} \leq \frac{\ell\epsilon}{\max(\ell, n^2)n} \leq \frac{\epsilon}{n}$. Hence Reed-Solomon Error correction algorithm will point $\overline{s_c}$ as a corrupted share, in which case P_c will be caught and his sharing will be ignored with probability $(1 - \frac{\epsilon}{n})$. Now since there

Figure 6.5: Protocol Upgrade1dto2d: Generates t-$(2d)$-sharing of ℓ secrets given t-$(1d)$-sharing of the same secrets.

$\mathsf{Upgrade1dto2d}(\mathcal{P}, \ell, ([s^1]_t, \ldots, [s^\ell]_t), \epsilon)$

1. All the parties invoke $\mathsf{Random}(\mathcal{P}, 1, \epsilon)$ to generate t-$(1d)$-sharing of a random secret s^0, i.e $[s^0]_t$. So party P_i has s_i^0, the i^{th} share of s^0.
2. Now every P_i invokes $\mathsf{VSS\text{-}Share}(P_i, \mathcal{P}, \ell+1, (s_i^0, s_i^1, \ldots, s_i^\ell), \frac{\epsilon}{n})$ to generate $[s_i^0]_t, [s_i^1]_t, \ldots, [s_i^\ell]_t$, where $s_i^0, s_i^1, \ldots, s_i^\ell$ are the i^{th} shares of secrets $s^0, s^1, \ldots, s^\ell$ respectively.
3. The parties in $\mathcal{P}$ jointly generate a random value r by invoking Protocol $\mathsf{RandomVector}(\mathcal{P})$.
4. Now to detect the parties P_k (at most t), who have generated $[\overline{s_k^0}]_t, [\overline{s_k^1}]_t, \ldots, [\overline{s_k^\ell}]_t$ such that $\overline{s_k^l} \neq s_k^l$ for some $l \in \{0, 1, \ldots, \ell\}$, all the parties publicly reconstruct $s_i = s_i^0 + \sum_{l=1}^{\ell} r^l s_i^l$ and $s = s^0 + \sum_{l=1}^{\ell} r^l s^l$ by executing following steps:
   (a) The parties in $\mathcal{P}$ compute $[s_i]_t = [s_i^0]_t + \sum_{l=1}^{\ell} r^l [s_i^l]_t$ and invoke $\mathsf{VSS\text{-}Rec}(\mathcal{P}, [s_i]_t)$ to publicly reconstruct s_i, for $i = 1, \ldots, n$.
   (b) Every party apply Reed-Solomon error correction algorithm (e.g. Berlekamp Welch Algorithm [119]) on $s_1, \ldots, s_n$, to recover s. Reed-Solomon error correction algorithm also points out the corrupted shares. Hence if s_i is pointed as a corrupted share, then $[s_i^0]_t, [s_i^1]_t, \ldots, [s_i^\ell]_t$ are ignored by every party.
5. Output $[[s^1]]_t, \ldots, [[s^\ell]]_t$.

---

are at most t corrupted parties who may generate wrong t-$(1d)$-sharings as above, the probability that all of them will be caught is $(1 - t\frac{\epsilon}{n}) \approx (1 - \epsilon)$.

Secrecy: It is easy to see that at any stage of the protocol, $\mathcal{A}_t$ learns not more than t shares for each $s^l, 1 \leq l \leq \ell$. Moreover, the publicly reconstructed value s (which is equal to $(s^0 + \sum_{l=1}^{\ell} r^l s^l)$) does not leak any information about the secrets. This is because the linear combination of the secrets are blinded by random s^0 and thus s will look completely random to $\mathcal{A}_t$. Hence all the secrets will be secure. □

Lemma 6.18 *Protocol Upgrade1dto2d has the following bounds:*

1. **Round Complexity:** *Eighteen Rounds.*
2. **Communication Complexity:** *Private and broadcast communication of $\mathcal{O}((\ell n^2 + n^4) \log \frac{1}{\epsilon})$ bits.*

PROOF: Step 1 and step 2 require eight rounds each (invokes VSS-Share). So Step 1 and step 2 can be completed in sixteen rounds. Step 3 requires five rounds. Step 3 has to be completed one round after step 2. Now since step 3 requires five rounds (invokes RandomVector), first four rounds of it can be executed in parallel with the last 4 rounds of step 2. Hence computation up to step 3 can be completed in seventeen rounds. Step 4 requires one round. Thus in total Upgrade1dto2d requires eighteen rounds. The communication complexity of Upgrade1dto2d can be verified easily. □

Remark 6.19 (Comparison with Existing Protocols) *In [12], the authors reported a protocol to upgrade t-$(1d)$-sharing to t-$(2d)$-sharing, where $n = 2t+1$. However, the protocol is non-robust. That is, if all the n parties behave honestly, then the protocol will perform the upgradation. Otherwise, the protocol will fail to do the upgradation, but will output a pair of parties, of which at least one is corrupted. On the other hand, our upgradation protocol is designed with $n = 3t+1$ and hence will always perform the upgradation successfully, irrespective of the behavior of the corrupted parties.*

6.7.2 An ABC protocol— Proving $c = ab$

Consider the following problem: let $P \in \mathcal{P}$ has properly generated $[a^1]_t, \ldots, [a^\ell]_t$ and $[b^1]_t, \ldots, [b^\ell]_t$. Now P wants to generate $[c^1]_t, \ldots, [c^\ell]_t$, where $c^l = a^l b^l$ for $l = 1, \ldots, \ell$. Moreover, during this process, an honest P does not want to leak any *additional* information about a^l, b^l and c^l. Furthermore, if P is corrupted, then he may intentionally fail to generate the above output in which case every body will know that party P is corrupted.

We propose a protocol called ProveCeqAB to achieve the above task. The protocol generates the correct output, except with error probability ϵ. The idea of the protocol is inspired from [48] with the following modification: we make use of our protocol VSS (instead of their statistical VSS protocol), which provides us with high efficiency, both in terms of communication and round complexity. In section 5.3 of previous chapter, a very similar idea has been presented for $n = 2t+1$ parties. Since here we have $n = 3t+1$ parties, few things can be simplified. Moreover, here we deal with t-$(1d)$-sharing of the values rather than $2d^\star$-sharing/$1d^\star$-sharing used in section 5.3 of previous chapter. For the sake of completeness and clarity, we discuss about the idea again with the simplifications in the context of $n = 3t+1$ parties.

We explain the idea of the protocol with a single pair (a, b). With respect to a single pair, the problem becomes like this: P has already t-$(1d)$-shared a and b. Now he wants to generate t-$(1d)$-sharing of c, where $c = ab$, without leaking any *additional* information about a, b and c. To achieve this goal, P first selects a random non-zero $\beta \in \mathbb{F}$ and generates t-$(1d)$-sharing of c, β and $d = \beta b$. All the parties in $\mathcal{P}$ then jointly generate a random value r and computes t-$(1d)$-sharing of $p = ra+\beta$ and reconstructs p from its t-$(1d)$-sharing. The parties then compute t-$(1d)$-sharing of $q = pb - d - rc$ and reconstruct q from its t-$(1d)$-sharing. Every party checks whether $q \stackrel{?}{=} 0$. If so then everybody accepts the t-$(1d)$-sharing of c as valid t-$(1d)$-sharing of ab. It is easy to check that q will be zero when P behaves honestly.

If a corrupted P shares $c \neq ab$, then the probability that $q = 0$ holds is negligible because of the random r. This can be argued as follows: $q = pb - d - rc = (ra+\beta)b - d - rc = rab - rc + \beta b - d = r(ab-c) + \beta b - d$. Now if P shares $c \neq ab$ and $d \neq \beta b$, then $q = r(ab-c) + \beta b - d$ will be non-zero, except for only one value of r. But since r is randomly generated, the probability that r is that value is $\frac{1}{|\mathbb{F}|}$ which is negligibly small. The secrecy follows from the fact that p is randomly distributed and $q = 0$. Protocol ProveCeqAB extends the above idea for ℓ pairs (a^l, b^l).

ProveCeqAB works on a field $\mathbb{F}$ which was used for protocol Random i.e $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq \max(\ell, n^2)n2^{-\kappa}$. This comes from the following facts: Since ProveCeqAB invokes Random with ϵ error probability, $\epsilon \geq \max(\ell, n^2)n2^{-\kappa}$ should hold. Moreover, ProveCeqAB invokes VSS-Share with ϵ error probability which enforces $\epsilon \geq \max(\ell, n^2)2^{-\kappa}$. Therefore, $\epsilon \geq \max(\ell, n^2)n2^{-\kappa}$ should hold for ProveCeqAB. Now the protocol is formally given in Fig. 6.6.

Figure 6.6: Protocol ProveCeqAB: An ABC Protocol for proving $c = ab$.

ProveCeqAB(P,$\mathcal{P}, \ell, [a^1]_t^P, [b^1]_t^P, \ldots, [a^\ell]_t^P, [b^\ell]_t^P$)

1. P chooses a random non-zero ℓ length tuple $(\beta^1, \ldots, \beta^\ell) \in \mathbb{F}^\ell$. In parallel, P invokes
   (a) VSS-Share$(P, \mathcal{P}, \ell, (c^1, \ldots, c^\ell), \epsilon)$ to generate t-$(1d)$-sharing of $(c^1, \ldots, c^\ell)$,
   (b) VSS-Share$(P, \mathcal{P}, \ell, (\beta^1, \ldots, \beta^\ell), \epsilon)$ to generate t-$(1d)$-sharing of $(\beta^1, \ldots, \beta^\ell)$,
   (c) VSS-Share$(P, \mathcal{P}, \ell, (d^1, \ldots, d^\ell), \epsilon)$ to generate t-$(1d)$-sharing of $(b^1\beta^1, \ldots, b^\ell\beta^\ell)$, where $d^l = b^l\beta^l$.

   If P is discarded in any of the three instances of VSS-Share, P fails to prove $c = ab$ and the protocol terminates here.

2. Now all the parties in $\mathcal{P}$ invoke RandomVector$(\mathcal{P})$ to generate a random value $r \in \mathbb{F}$.

3. For every $l \in \{1, \ldots, \ell\}$, all parties locally compute $[p^l]_t = (r[a^l]_t^P + [\beta^l]_t^P)$ and invoke VSS-Rec$(\mathcal{P}, [p^l]_t)$ to reconstruct p^l.

4. For every $l \in \{1, \ldots, \ell\}$, the parties locally compute $[q^l]_t = \big(p^l[b^l]_t^P - [d^l]_t^P - r[c^l]_t^P\big)$ and invoke VSS-Rec$(\mathcal{P}, [q^l]_t)$ to reconstruct q^l.

5. The parties then check $q^l \stackrel{?}{=} 0$. If not then every party concludes that P fails to prove $c = ab$ and the protocol terminates here. Otherwise P has proved that $c = ab$.

Lemma 6.20 *Protocol ProveCeqAB satisfies the following properties:*

1. **Correctness:** *If P is honest, then except with probability ϵ, P will be able to generate $[c^1]_t^P, \ldots, [c^\ell]_t^P$. If P is corrupted and the protocol succeeds then except with probability ϵ, P has generated $[c^1]_t^P, \ldots, [c^\ell]_t^P$, where $c^l = a^lb^l$, for $l = 1, \ldots, \ell$.*

2. **Secrecy:** *If P is honest then a^l, b^l, c^l will be information theoretically secure for all $l = 1, \ldots, \ell$.*

PROOF: **Correctness**: We show that if P is honest, then P will generate $[c^1]_t^P, \ldots, [c^\ell]_t^P$, except with probability ϵ. If P is honest, then except with error probability ϵ, he will not be discarded in any of the three instances of $\mathsf{VSS\text{-}Share}$. The parties will jointly generate r, except with probability ϵ. After this, it is clear that an honest party P will never fail to prove $c = ab$ as $q^l = 0$ holds for all l.

Now we show that if a corrupted P has generated $[c^1]_t^P, \ldots, [c^\ell]_t^P$, then except with probability ϵ, $c^l = a^l b^l$ for $l = 1, \ldots, \ell$. If a corrupted P has proved $c = ab$, then it must be the case that $q^l = 0$ for all $l = 1, \ldots, \ell$. Now $q^l = p^l b^l - d^l - rc^l = (ra^l + \beta^l)b^l - d^l - rc^l = ra^l b^l - rc^l + \beta^l b^l - d^l = r(a^l b^l - c^l) + \beta^l b^l - d^l$. Now $q^l = 0$ for $l = 1, \ldots, \ell$ will hold when one of the following three cases happens.

1. **P shares $c^l = a^l b^l$ and $d^l = \beta^l b^l$:** If this is the case then $c^l = a^l b^l$ for $l = 1, \ldots, \ell$ without any error probability.

2. **P shares $c^l \neq a^l b^l$ and $d^l = \beta^l b^l$ and $r = 0$:** If this is the case, then $c^l = a^l b^l$ for $l = 1, \ldots, \ell$, except with error probability ϵ. This is because r is generated randomly and therefore the probability that $r = 0$ is $\frac{1}{|\mathbb{F}|} \leq \frac{\epsilon}{\max(\ell, n^2)n} \leq \epsilon$. It is easy to see that q^l will be non-zero in this case, except when $r = 0$.

3. **P shares $c^l \neq a^l b^l$ and $d^l \neq \beta^l b^l$ and r has a specific value:** If this is the case, then $c^l = a^l b^l$ for $l = 1, \ldots, \ell$, except with error probability ϵ. This is because, there is only one specific value of r for which the value $q^l = r(a^l b^l - c^l) + \beta^l b^l - d^l$ will be zero even though $c^l \neq a^l b^l$ and $d^l \neq \beta^l b^l$. We prove this by contradiction. Let there are two unequal values r_1 and r_2 such that $r_1(a^l b^l - c^l) + \beta^l b^l - d^l = 0$ and $r_2(a^l b^l - c^l) + \beta^l b^l - d^l = 0$ holds even though $c^l \neq a^l b^l$ and $d^l \neq \beta^l b^l$. This implies that

$$\begin{aligned} r_1(a^l b^l - c^l) + \beta^l b^l - d^l &= r_2(a^l b^l - c^l) + \beta^l b^l - d^l \\ \Rightarrow r_1(a^l b^l - c^l) &= r_2(a^l b^l - c^l) \\ \Rightarrow r_1 &= r_2, \quad \text{which is a contradiction to our assumption.} \end{aligned}$$

   Hence there is only one value for r for which $q^l = r(a^l b^l - c^l) + \beta^l b^l - d^l$ will be zero. But since r is randomly generated, the probability that r is that value is $\frac{1}{|\mathbb{F}|} \leq \frac{\epsilon}{\max(\ell, n^2)n} \leq \epsilon$.

The above cases show that if $q^l = 0$, then $c^l = a^l b^l$, except with probability ϵ. Therefore if P has generated $[c^1]_t^P, \ldots, [c^\ell]_t^P$, then except with probability ϵ, $c^l = a^l b^l$ for $l = 1, \ldots, \ell$.

Secrecy: We now prove the secrecy of a^l, b^l, c^l for all $l = 1, \ldots, \ell$ when P is honest. From the secrecy property of $\mathsf{VSS\text{-}Share}$ and property of t-$(1d)$-sharing, a^l, b^l and c^l will remain secure. Now we will show that both p^l and q^l will not leak any information about a^l, b^l and c^l. Clearly $p^l = (ra^l + \beta^l)$ will look completely random to the adversary as β^l is randomly chosen. Furthermore $q^l = 0$ and hence q^l does not leak any information on a^l, b^l and c^l. Hence the lemma. □

Lemma 6.21 *Protocol* $\mathsf{ProveCeqAB}$ *achieves the following:*

1. **Round Complexity:** *Ten rounds.*

2. **Communication Complexity:** *Private and Broadcast communication of $\mathcal{O}((\ell n + n^3)\log\frac{1}{\epsilon})$ bits.*

PROOF: Step 1 requires eight rounds (invokes three instances of VSS-Share in parallel). Step 2 requires five rounds (invokes one instance of RandomVector). Detailed observation confirms that step 1 and step 2 can be executed in parallel. So step 1 and 2 require eight rounds in total. Step 3 and 4 require one round each (invokes several instances of VSS-Rec). Hence in total ProveCeqAB requires ten rounds.

The communication complexity can be verified easily. □

6.7.3 Our Multiplication Protocol

Finally, we present a multiplication protocol, called Mult which allows the parties to generate $[c^1]_t, \ldots, [c^\ell]_t$ given $[a^1]_t, \ldots, [a^\ell]_t$ and $[b^1]_t, \ldots, [b^\ell]_t$, where a^l's and b^l's are random and $c^l = a^l b^l$ for $l = 1, \ldots, \ell$. For simplicity, we first explain the idea of the protocol for a single triple $[a]_t, [b]_t$ and $[c]_t$.

Given $[a]_t, [b]_t$, parties first invoke Upgrade1dto2d to generate $[[a]]_t$ and $[[b]]_t$. Then every party P_i computes $a_i b_i$ and generates $[a_i b_i]_t^{P_i}$ by executing ProveCeqAB (though the corrupted parties may fail to generate $[a_i b_i]_t^{P_i}$), where a_i and b_i are the i^{th} shares of a and b. Since $a_1 b_1, \ldots, a_n b_n$ are n points on a $2t$ degree polynomial, say $C(x)$, whose constant term is c, by Lagrange interpolation formula [46], c can be computed as $c = \sum_{i=1}^{n} r_i(a_i b_i)$ where $r_i = \prod_{j=1, j\neq i}^{n} \frac{-j}{i-j}$. The vector $(r_1, \ldots, r_n)$ is called recombination vector [46] which is public and known to every party. So we write $c = Lagrange(a_1 b_1, \ldots, a_n b_n) = \sum_{i=1}^{n} r_i(a_i b_i)$. Now all parties compute $[c]_t = Lagrange([a_1 b_1]_t, \ldots, [a_n b_n]_t) = \sum_{i=1}^{n} r_i[a_i b_i]_t$, to obtain the desired output. Notice that since $C(x)$ is of degree $2t$, we need $2t+1$ parties to successfully generate $a_i b_i$ value (a $2t$ degree polynomial requires $2t+1$ points on it to be interpolated correctly). So, even if t corrupted parties fail to generate $[a_i b_i]_t$, our protocol will work. Our protocol Mult follows the above technique for ℓ pairs simultaneously. Our protocol is motivated from the protocol of [48].

Mult works on a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq \max(\ell, n^2)n^2 2^{-\kappa}$. This is because Mult invokes ProveCeqAB with $\frac{\epsilon}{n}$ error probability and from previous section, ProveCeqAB requires $\epsilon \geq \max(\ell, n^2)n2^{-\kappa}$ to bound its error probability by ϵ. Now the protocol is formally given in Fig. 6.7.

Lemma 6.22 *Protocol Mult satisfies the following properties:*

1. **Correctness***: Except with probability ϵ, the protocol correctly outputs $([c^1]_t, \ldots, [c^\ell]_t)$, given $([a^1]_t, \ldots, [a^\ell]_t)$ and $([b^1]_t, \ldots, [b^\ell]_t)$. Moreover, for $l = 1, \ldots, \ell$, $c^l = a^l b^l$.*

2. **Secrecy***: The adversary will have no information about (a^k, b^k, c^k), for $k = 1, \ldots, \ell$.*

PROOF: **Correctness:** Both the instances of Upgrade1dto2d will successfully generate their outputs, except with probability ϵ. By Lemma 6.20, every honest party P_i will generate $[c_i^1]_t, \ldots, [c_i^\ell]_t$, except with probability $\frac{\epsilon}{n}$. Therefore all the honest P_i's will generate $[c_i^1]_t, \ldots, [c_i^\ell]_t$, except with probability $(2t+1)\frac{\epsilon}{n} \approx \epsilon$. Moreover, by Lemma 6.20, a corrupted party P_i who generated $[c_i^l]_t$ has indeed

Figure 6.7: Protocol Mult: Generates $[c^l]_t$ from $[a^l]_t$ and $[b^l]_t$ for $l = 1, \ldots, \ell$.

Mult$(\mathcal{P}, \ell, ([a^1]_t, [b^1]_t), \ldots, ([a^\ell]_t, [b^\ell]_t), \epsilon)$

1. All the parties invoke
   (a) Upgrade1dto2d$(\mathcal{P}, \ell, ([a^1]_t, \ldots, [a^\ell]_t), \epsilon)$ to generate $[[a^1]]_t, \ldots, [[a^\ell]]_t$.
   (b) Upgrade1dto2d$(\mathcal{P}, \ell, ([b^1]_t, \ldots, [b^\ell]_t), \epsilon)$ to generate $[[b^1]]_t, \ldots, [[b^\ell]]_t$.
2. Each party P_i invokes ProveCeqAB$(P_i, \mathcal{P}, \ell, [a_i^1]_t, [b_i^1]_t, \ldots, [a_i^\ell]_t, [b_i^\ell]_t, \frac{\epsilon}{n})$ to produce $[c_i^1]_t, \ldots, [c_i^\ell]_t$ such that $c_i^l = a_i^l b_i^l$ for $l = 1, \ldots, \ell$ where a_i^l and b_i^l are the i^{th} shares of a^l and b^l. At most t (corrupted) parties may fail to execute ProveCeqAB. For simplicity assume first $2t + 1$ parties are successful in executing ProveCeqAB.
3. Now for each $l \in \{1, \ldots, \ell\}$, first $(2t + 1)$ parties have produced $[c_1^l]_t, \ldots, [c_{(2t+1)}^l]_t$. So for $l = 1, \ldots, \ell$, parties in $\mathcal{P}$ compute $[c^l]_t$ as follows: $[c^l]_t = Lagrange([c_1^l]_t, \ldots, [c_{2t+1}^{(l)}]_t)$.

shared $c_i^l = a_i^l b_i^l$ for all l, except with probability $\frac{\epsilon}{n}$. Since there can be at most t corrupted P_i's, the probability that all the corrupted parties who generated $[c_i^l]_t$ have indeed shared $c_i^l = a_i^l b_i^l$ for all l, is at least $(1 - \frac{\epsilon}{n})^t \approx (1 - t\frac{\epsilon}{n}) \approx (1 - \epsilon)$. Hence $[c^1]_t, \ldots, [c^\ell]_t)$ are generated correctly, except with probability ϵ.

Secrecy: Now according to the secrecy of protocol ProveCeqAB, $(c_i^1, \ldots, c_i^\ell)$, $(a_i^1, \ldots, a_i^\ell)$ and $(b_i^1, \ldots, b_i^\ell)$ will remain secure for every honest P_i. Now since $[c^1]_t, \ldots, [c^\ell]_t$ is generated by taking linear combination of $[c_i^1]_t^{P_i}, \ldots, [c_i^\ell]_t^{P_i}$'s (in which at least $t + 1$ set of $c_i^1, \ldots, c_i^\ell$ are unknown to $\mathcal{A}_t$), the secrecy of $c^1, \ldots, c^\ell$ is guaranteed. □

Lemma 6.23 *Protocol Mult has the following bounds:*

1. **Round Complexity:** *Twenty eight rounds.*
2. **Communication Complexity:** *Private and broadcast communication of* $\mathcal{O}((\ell n^2 + n^4) \log \frac{1}{\epsilon})$ *bits.*

PROOF: Step 1 requires eighteen rounds (invokes two instances of Upgrade1dto2d in parallel). Step 2 requires ten rounds (invokes n parallel instances of ProveCeqAB). In total Mult requires twenty eight rounds. The communication complexity of Mult can be verified easily. □

6.8 Statistical MPSI Protocol with $n = 3t + 1$

We now present our statistical MPSI protocol with $n = 3t+1$. Our MPSI protocol works on a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq \max(m^2, n)n^3 2^{-\kappa}$. This follows from the fact that in the Computation Phase of our MPSI protocol, Mult is invoked with ϵ probability and $\ell = n(m + 1)^2$. The above relation between ϵ and κ also reflects the condition put by other sub-protocols such as Random and VSS-Share (invoked in Preparation Phase and Input Phase, respectively). Thus each field element from $\mathbb{F}$ can be represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

We first present the protocol for Input Phase and Preparation Phase, where t-($1d$)-sharing of the coefficients of $r^i(x)$ and $f^i(x)$ polynomials are generated.

Figure 6.8: Input and Preparation Phase of our statistical MPSI Protocol

Input Phase

1. Every $P_i \in \mathcal{P}$ represents his set $S_i = \{e_i^1, \ldots, e_i^m\}$ by a polynomial $f^i(x)$ of degree m such that $f^i(x) = (x - e_i^1) \cdots (x - e_i^m) = a^{(0,i)} + a^{(1,i)}x + \ldots + a^{(m,i)}x^m$.

2. P_i then invokes VSS-Share$(P_i, \mathcal{P}, m, (a^{(0,i)}, \ldots, a^{(m-1,i)}), \frac{\epsilon}{n})$ to generate $[a^{(0,i)}]_t, \ldots, [a^{(m-1,i)}]_t$. If P_i is discarded then t-($1d$)-sharing of m default values are assumed as P_i's input. Moreover, since $a^{(m,i)} = 1$ always, every party in $\mathcal{P}$ assumes a predefined t-($1d$)-sharing for 1, namely $[1]_t$ on behalf of $a^{(m,i)}$ (see Remark 6.24).

Preparation Phase:

1. The parties in $\mathcal{P}$ invoke Random$(\mathcal{P}, n(m+1), \epsilon)$ to generate t-($1d$)-sharings of $n(m+1)$ values denoted by $[b^{(0,i)}]_t, \ldots, [b^{(m,i)}]_t$ for $i = 1, \ldots, n$.

2. Now the parties assume that $r^i(x) = b^{(0,i)} + b^{(1,i)}x + \ldots + b^{(m,i)}x^m$ for $i = 1, \ldots, n$.

Remark 6.24 *In any MPSI protocol that computes the intersection of the sets of the parties using the function given in (6.1), $\mathcal{A}_t$ may disrupt the security of the protocol by forcing a corrupted party to input a zero polynomial representing his set. This is because $\mathcal{A}_t$ will then come to know the intersection of the sets of the remaining parties at the end of computation of the protocol [116, 113]. So to stop a corrupted party to input a zero polynomial, the authors of [116, 113] specified the following trick. They have noticed that the coefficient of m^{th} degree term in every P_j's polynomial $f^j(x) = \prod_{k=1}^m (x - e_j^k)$ is 1 always. Hence, every party assumes a predefined $[1]_t$ on behalf of the m^{th} coefficient of $f^j(x)$ polynomial of every party (instead of allowing individual parties to t-($1d$)-share the m^{th} coefficient of their $f^j(x)$ polynomial). This stops the corrupted parties to commit a zero polynomial.*

Lemma 6.25 *The protocol for Input and Preparation Phase satisfies the following properties:*

1. **Correctness**: *Except with probability ϵ, Input Phase and Preparation Phase produces correct t-($1d$)-sharing for the coefficients of polynomials $f^i(x)$ and $r^i(x)$ for all $i = 1, \ldots, n$.*

2. **Secrecy**: *All the coefficients of $f^i(x)$ such that P_i is honest and all the coefficients of $r^i(x)$ for all $i = 1, \ldots, n$ remain unknown to $\mathcal{A}_t$.*

PROOF: **Correctness:** We first show that Input Phase will generate its correct output, except with probability ϵ. An honest party P_i will correctly generate t-$(1d)$-sharing of the coefficients of his polynomial $f^i(x)$ without being discarded, with probability at least $(1 - \frac{\epsilon}{n})$. Therefore the probability that all the $2t + 1$ honest parties will correctly generate t-$(1d)$-sharing of the coefficients of their polynomial without being discarded, is at least $(1 - (2t + 1)\frac{\epsilon}{n}) \approx (1 - \epsilon)$.

Moreover if a corrupted P_i has generated t-$(1d)$-sharing of m values (which are supposed to be m coefficients of his input polynomial), then those sharing are correct, with probability at least $(1 - \frac{\epsilon}{n})$. Therefore the probability that the sharings generated by all the corrupted parties are correct is at least $(1 - \epsilon)$.

It is easy to see that Preparation Phase has an error probability of ϵ (because Random has been invoked with error probability ϵ).

Secrecy: Secrecy follows from the secrecy property of VSS-Share and Random protocol. □

Lemma 6.26 *Input and Preparation Phase has the following bounds:*

1. **Round Complexity:** *Eight rounds.*
2. **Communication Complexity:** *Private and Broadcast communication of $\mathcal{O}((mn^3 + n^4)\log\frac{1}{\epsilon})$ bits.*

PROOF: Input Phase and Preparation Phase can be executed in parallel. Since both of them requires eight rounds, in total Input Phase and Preparation Phase require eight rounds. The communication complexity can be obtained from the communication complexity of VSS-Share and Random by putting appropriate value of ℓ. □

After input and preparation phase, in the Computation Phase (given in Fig. 6.9) the parties jointly compute $F(x) = \sum_{i=1}^{n} r^i(x)f^i(x)$ such that the coefficients of $F(x)$ are t-$(1d)$-shared. In Output Phase, the coefficients of $F(x)$ are publicly reconstructed. Then each party locally evaluates $F(x)$ at each element of his private set. All the elements at which $F(x) = 0$ belongs to the intersection of the n sets, with very high probability.

Lemma 6.27 *Given that Input Phase and Preparation Phase generate correct outputs, Computation Phase correctly outputs t-$(1d)$-sharing of the coefficients of $F(x)$ and Output Phase correctly reconstructs the coefficients of $F(x)$ publicly, except with probability ϵ.*

PROOF: Follows from **Correctness** of Mult and VSS-Rec. □

Lemma 6.28 *Computation Phase and Output Phase achieves the following bounds:*

1. **Round Complexity**: *Twenty Nine rounds in total.*
2. **Communication Complexity:** *Private and broadcast communication of $\mathcal{O}((m^2n^3 + n^5)\log\frac{1}{\epsilon})$ bits.*

PROOF: Computation Phase requires twenty eight rounds and Output Phase requires one round. Thus in total both the phases require twenty nine rounds. Communication complexity can be verified easily. □

Figure 6.9: Protocol for $\mathsf{Computation\ Phase}$ and $\mathsf{Output\ Phase}$ of our MPSI protocol.

Computation Phase

1. Let $F^i(x) = r^i(x)f^i(x) = c^{(0,i)} + c^{(1,i)}x + \ldots + c^{(2m,i)}x^{2m}$ for $i = 1, \ldots, n$. For $i = 1, \ldots, n$, to generate $[c^{(0,i)}]_t, \ldots, [c^{(2m,i)}]_t$, the parties in $\mathcal{P}$ have to first multiply the sharings of the coefficients of $r^i(x)$ and $f^i(x)$ and then they have to perform appropriate additions. It is achieved in the following way:
   (a) To generate $[a^{(0,i)}b^{(0,i)}]_t, [a^{(0,i)}b^{(1,i)}]_t, \ldots, [a^{(m,i)}b^{(m-1,i)}]_t, [a^{(m,i)}b^{(m,i)}]_t$ (i.e the pairwise coefficient multiplication of $r^i(x)$ and $f^i(x)$) for all i, the parties invoke only one instance of $\mathsf{Mult}(\mathcal{P}, n(m+1)^2, ([a^{(0,1)}]_t, [b^{(0,1)}]_t), ([a^{(0,1)}]_t, [b^{(1,1)}]_t), \ldots, ([a^{(m,1)}]_t, [b^{(m,1)}]_t), \ldots, ([a^{(m,n)}]_t, [b^{(m,n)}]_t), \epsilon)$ with $\ell = n(m+1)^2$ (every coefficient of $r^i(x)$ should be multiplied with every coefficient of $f^i(x)$ for all i).
   (b) The parties compute the following for all $i = 1, \ldots, n$.
      - $[c^{(0,i)}]_t = [a^{(0,i)}b^{(0,i)}]_t$,
      - $[c^{(1,i)}]_t = [a^{(0,i)}b^{(1,i)}]_t + [a^{(1,i)}b^{(0,i)}]_t$,
      - $[c^{(2,i)}]_t = [a^{(0,i)}b^{(2,i)}]_t + [a^{(2,i)}b^{(0,i)}]_t + [a^{(1,i)}b^{(1,i)}]_t$,
      - $\ldots$,
      - $[c^{(2m,i)}]_t = [a^{(m,i)}b^{(m,i)}]_t$.
2. Let $F(x) = \sum_{i=1}^{n} F^i(x) = d^{(0)} + d^1x + \ldots + d^{2m}x^{2m}$. To generate $[d^0]_t, \ldots, [d^{2m}]_t$, the parties compute $[d^j]_t = \sum_{i=1}^{n}[c^{(j,i)}]_t$ for $j = 0, \ldots, 2m$.

Output Phase

1. The parties invoke $\mathsf{VSS\text{-}Rec}(\mathcal{P}, [d^j]_t)$ to publicly reconstruct d^j for $j = 0, \ldots, 2m$. Thus the parties have reconstructed $F(x)$ whose coefficients are d^j for $j = 0, \ldots, 2m$.
2. Each P_i with his private set $S_i = \{e_i^1, \ldots, e_i^m\}$ locally checks whether $F(e_i^k) \stackrel{?}{=} 0$ for $k = 1, \ldots, m$. If $F(e_i^k) = 0$, then P_i adds e_i^k in a set I_i (initially $I_i = \emptyset$). P_i outputs I_i as the intersection set $S_1 \cap S_2 \ldots, \cap S_n$.

We now show that $F(x)$ (of (6.1)) does not leak any information other than what can be derived from the inputs and outputs of the t corrupted parties. The proof is taken from [113]. We also show that every honest party learns correct $I = S_1 \cap \ldots \cap S_n$ from $F(x)$, except with probability at most ϵ. A very basic outline of this proof is given in [113].

Lemma 6.29 *$F(x)$ of (6.1) does not leak any extra information to $\mathcal{A}_t$, other than what can be inferred by the data sets of the corrupted parties and the intersection of the data sets of all the parties.*

PROOF: Let I be the intersection of the data sets of all the parties. Recall that $F(x) = \sum_{i=1}^{n} f^i(x)r^i(x)$. Now we can write $F(x)$ as $F(x) = \prod_{a \in I}(x-a)E(x)$. Here $E(x) = \sum_{i=1}^{n} g^i(x)r^i(x)$ where $g^i(x) = \prod_{e_i^k \notin I}(x - e_i^k)$. It is easy to see that $gcd(g^1(x), \ldots, g^n(x)) = 1$ and $deg(r^i(x)) \geq deg(g^j(x))$ for all $i, j = 1, \ldots, n$. In fact $deg(r^i(x)) = m$ and $deg(g^i(x)) \leq m$ for all $i = 1, \ldots, n$, where m is the size of the data set of all the parties. We will now show that the polynomial $E(x)$ will be randomly distributed over $\mathbb{F}$ and therefore by learning $F(x)$, the adversary $\mathcal{A}_t$ will learn no extra information (regarding the private data-set of the honest parties), other than the intersection of the data sets of all the parties i.e I. We prove this in the Lemma 6.31. $\square$

Before proving Lemma 6.31, we prove the following lemma:

Lemma 6.30 *Let $g^1(x)$ and $g^2(x)$ be two polynomials over $\mathbb{F}$ of same degree, such that $gcd(g^1(x), g^2(x)) = 1$ and $deg(g^i(x)) \leq m$ for all $i = 1, 2$. Let $r^1(x)$ and $r^2(x)$ be two random polynomials over $\mathbb{F}$ such that $deg(r^i(x)) = m$ for all $i = 1, 2$. Then the polynomial $E(x) = \sum_{i=1}^{2} g^i(x)r^i(x)$ will be randomly distributed over $\mathbb{F}$.*

PROOF: Let $g^1(x)$ and $g^2(x)$ be polynomials over $\mathbb{F}$ of same degree, such that $gcd(g^1(x), g^2(x)) = 1$. Let $r^1(x)$ and $r^2(x)$ be two random polynomials over $\mathbb{F}$ of same degree such that $deg(r^i(x)) \geq deg(g^j(x))$, for all $i, j = 1, 2$. Moreover, $deg(r^i(x)) = m$ and $deg(g^i(x)) \leq m$, for $i = 1, 2$. Furthermore, let $E(x) = g^1(x)r^1(x) + g^2r^2(x)$ and $deg(g^i(x)) = \alpha$. Evidently $\alpha \leq m$.

The outline of the proof is as follows:

1. Given any fixed polynomials $g^1(x), g^2(x)$ and $E(x)$, satisfying the above given conditions, we first compute the number of $(r^1(x), r^2(x))$ pairs z over $\mathbb{F}$, such that $g^1(x)r^1(x) + g^2(x)r^2(x) = E(x)$.

2. We next show that given any fixed polynomials $g^1(x)$ and $g^2(x)$, the total number of all possible $(r^1(x), r^2(x))$ pairs over $\mathbb{F}$, divided by z, is equal to the number of all possible polynomials $E(x)$ over $\mathbb{F}$. This implies that, if $gcd(g^1(x), g^2(x)) = 1$ and we choose the coefficients of $r^1(x), r^2(x)$ uniformly and independently from $\mathbb{F}$, then coefficients of the result polynomial $E(x)$ are distributed uniformly and independently over $\mathbb{F}$.

We now first determine the value of z, which is the number of $(r^1(x), r^2(x))$ pairs over $\mathbb{F}$, such that $g^1(x)r^1(x) + g^2(x)r^2(x) = E(x)$, for given $g^1(x), g^2(x)$ and $E(x)$. Let us assume that for this particular $E(x)$, there exists at least one pair $\overline{r^1}(x)$, $\overline{r^2}(x)$ such that $g^1(x)\overline{r^1}(x) + g^2(x)\overline{r^2}(x) = E(x)$. If there is another pair pair $\widehat{r^1}(x), \widehat{r^2}(x)$ such that $g^1(x)\widehat{r^1}(x) + g^2(x)\widehat{r^2}(x) = E(x)$, then we have

$$\begin{aligned} g^1(x)\overline{r^1}(x) + g^2(x)\overline{r^2}(x) &= g^1(x)\widehat{r^1}(x) + g^2(x)\widehat{r^2}(x) \\ \Longrightarrow g^1(x)(\overline{r^1}(x) - \widehat{r^1}(x)) &= g^2(x)(\widehat{r^2}(x) - \overline{r^2}(x)). \end{aligned}$$

As $gcd(g^1(x), g^2(x)) = 1$, we may conclude that $g^1(x)|(\widehat{r^2}(x) - \overline{r^2}(x))$ (which means $g^1(x)$ divides $(\widehat{r^2}(x) - \overline{r^2}(x)))$ and $g^2(x)|(\overline{r^1}(x) - \widehat{r^1}(x))$. Let $p(x)$ be a polynomial such that $p(x) \cdot g^1(x) = (\widehat{r^2}(x) - \overline{r^2}(x))$ and $p(x) \cdot g^2(x) = (\overline{r^1}(x) - \widehat{r^1}(x))$. Notice that the polynomial $p(x)$ will be of degree $m - \alpha$, where $\alpha = deg(g^1(x)) = deg(g^2(x))$.

We next show the following for $p(x)$:

1. We first show that each choice of polynomial $p(x)$ of degree $m - \alpha$, determines exactly one unique pair $(\widehat{r^1}(x), \widehat{r^2}(x))$. such that $g^1(x)\widehat{r^1}(x) + g^2(x)\widehat{r^2}(x) = E(x)$;

2. We next show that there exist no pairs $(\widehat{r^1}(x), \widehat{r^2}(x))$ which is not determined by any choice of polynomial $p(x)$ of degree $m-\alpha$, such that $g^1\widehat{r^1}+g^2\widehat{r^2}=E$.

So we first show that each choice of polynomial $p(x)$ of degree $m-\alpha$, determines exactly one unique pair $\widehat{r^1}(x), \widehat{r^2}(x)$ such that $g^1(x)\widehat{r^1}(x)+g^2(x)\widehat{r^2}(x)=E(x)$. Recall that by the property of $p(x)$, we have $\widehat{r^1}(x)=\overline{r^1}(x)-g^2(x)p(x)$ and $\widehat{r^2}(x)=\overline{r^2}(x)+g^1(x)p(x)$. Now as we have fixed $g^1(x), g^2(x), \overline{r^1}(x)$ and $\overline{r^2}(x)$, a choice for $p(x)$ will determine both $\widehat{r^1}(x)$ and $\widehat{r^2}(x)$. Moreover, if these assignments were not unique, then there would exist another polynomial $p'(x)$ of degree $m-\alpha$ such that either $\widehat{r^1}(x)=\overline{r^1}(x)-g^2(x)p(x)=\widehat{r^1}(x)=\overline{r^1}(x)-g^2(x)p'(x)$ or $\widehat{r^2}(x)=\overline{r^1}(x)+g^1(x)p(x)=\overline{r^1}(x)+g^1(x)p'(x)$. These conditions further imply that either $g^2(x)p(x)=g^2(x)p'(x)$ or $g^1(x)p(x)=g^1(x)p'(x)$ for some polynomials $p(x), p'(x)$, where $p(x)\neq p'(x)$. But this is impossible.

We next show that there exist no pair $(\widehat{r^1}(x), \widehat{r^2}(x))$ such that $g^1(x)\widehat{r^1}(x)+g^2(x)\widehat{r^2}(x)=E(x)$, but still $(\widehat{r^1}(x), \widehat{r^2}(x))$ is not determined by any choice of the polynomial $p(x)$ of degree $m-\alpha$. So let for different polynomials $p(x), p'(x)$ of degree $m-\alpha$, we have $p'(x)g^2(x)=\overline{r^1}(x)-\widehat{r^1}(x)$ and $p(x)g^1(x)=\widehat{r^2}(x)-\overline{r^2}(x)$. As we proved that $g^2(x)|(\overline{r^1}(x)-\widehat{r^1}(x))$ and $g^1(x)|(\widehat{r^2}(x)-\overline{r^2}(x))$, we can represent $g^1(x)$ and $g^2(x)$ in the above fashion without any loss of generality. Then this implies that

$$\begin{aligned} g^1(x)(\overline{r^1}(x)-\widehat{r^1}(x)) &= g^2(x)(\widehat{r^2}(x)-\overline{r^2}(x)) \\ \Longrightarrow g^1(x)(p'(x)g^2(x)) &= g^2(x)(p(x)g^1(x)) \\ \Longrightarrow p(x) &= p'(x) \quad \text{which is a contradiction.} \end{aligned}$$

This implies that there exist no pair $(\widehat{r^1}(x), \widehat{r^2}(x))$ such that $g^1(x)\widehat{r^1}(x)+g^2(x)\widehat{r^2}(x)=E(x)$, but still $(\widehat{r^1}(x), \widehat{r^2}(x))$ is not determined by any choice of the polynomial $p(x)$ of degree $m-\alpha$.

From the above discussion, we find that the number of polynomials $p(x)$ of degree $m-\alpha$ over $\mathbb{F}$, is exactly equal to the number of $(r^1(x), r^2(x))$ pairs such that $g^1(x)r^1(x)+g^2(x)r^2(x)=E(x)$. As there are $|\mathbb{F}|^{m-\alpha+1}$ such polynomials $p(x)$, we have $z=|\mathbb{F}|^{m-\alpha+1}$.

We now show that the total number of $(r^1(x), r^2(x))$ pairs over $\mathbb{F}$, divided by z, is equal to the total number of polynomials of degree $m+\alpha+1$ over $\mathbb{F}$. There are total $|\mathbb{F}|^{2m+2}$ possible $(r^1(x), r^2(x))$ pairs over $\mathbb{F}$. Now $\frac{|\mathbb{F}|^{2m+2}}{z}=\frac{|\mathbb{F}|^{2m+2}}{|\mathbb{F}|^{m-\alpha+1}}=|\mathbb{F}|^{m+\alpha+1}$. But $|\mathbb{F}|^{m+\alpha+1}$ denotes the total number of possible $E(x)$ polynomials over $\mathbb{F}$. What this shows is the following: suppose we fix $g^1(x)$ and $g^2(x)$. Then $\mathbb{F}^{2m+2}$ denotes the total space of pair of polynomials, each of degree m over $\mathbb{F}$. There will be $|\mathbb{F}|^{m-\alpha+1}$ pair of polynomials in this total space, which will determine a specific $E(x)$ polynomial. Like this, there can be $\frac{|\mathbb{F}|^{2m+2}}{|\mathbb{F}|^{m-\alpha+1}}=|\mathbb{F}|^{m+\alpha+1}$ distinct $E(x)$ polynomials which are possible by different choice of $(r^1(x), r^2(x))$. This prove that $E(x)$ is a random polynomial over $\mathbb{F}$, because $E(x)$ is a polynomial of degree $m+\alpha$ over $\mathbb{F}$ and $r^1(x), r^2(x)$ are random polynomials. $\square$

Now extending the above lemma for the case of n polynomials in a straight forward way, we get the following lemma:

Lemma 6.31 *Let there are n polynomials $g^1(x), \ldots, g^n(x)$ over $\mathbb{F}$ such that $gcd(g^1(x), \ldots, g^n(x)) = 1$. Let there are n randomly chosen polynomials $r^1(x), \ldots, r^n(x)$ such that $deg(r^i(x)) \geq deg(g^j(x))$ for all $i, j = 1, \ldots, n$. Moreover, $deg(r^i(x)) = m$ and $deg(g^i(x)) \leq m$ for all $i = 1, \ldots, n$. Then the polynomial $E(x) = \sum_{i=1}^{n} g^i(x)r^i(x)$ will be randomly distributed over $\mathbb{F}$.*

We next show that computing the intersection of the n sets by computing the function in (6.1) will give the correct output, except with error probability ϵ.

Lemma 6.32 *Let $F(x) = f^1(x)r^1(x) + \ldots + f^n(x)r^n(x)$, where $f^i(x)$ is an m degree polynomial representing the data set S_i and $r^i(x)$ is a completely random polynomial of degree m, for $i = 1, \ldots, n$. Then every honest party learns correct $I = S_1 \cap \ldots \cap S_n$ from $F(x)$, except with error probability at most ϵ.*

Proof: We show that with very high probability, erroneous elements (which does not belong to the intersection of the n sets) are not inserted into the intersection set I of an honest party. From the proof of the previous lemma, we have $F(x) = \sum_{i=1}^{n} f^i(x)r^i(x) = \prod_{a \in I}(x-a)E(x)$, where the coefficients of $E(x)$ are randomly distributed over $\mathbb{F}^{m+\alpha+1} = \mathbb{F}^{2m-|I|+1}$. The polynomial $E(x)$ has degree $2m - |I|$ and therefore it has same number of roots. Let P be the union of the private data-sets of the honest parties. As there are $2t+1$ honest parties, it implies that $|P| \leq (2t+1)m$. Now an erroneous element e from $\mathbb{F}$ will enter into I of some honest party, if e is the root of $E(x)$ and simultaneously belongs to P.

We estimate the error probability of the above event crudely and show that the error probability is negligible. An element $e \in \mathbb{F}$ is a root of $E(x)$ with probability $\frac{2m-|I|}{|\mathbb{F}|}$. Moreover, $e \in \mathbb{F}$ belongs to P with probability at most $\frac{|P|}{|\mathbb{F}|}$. Therefore, e can be in I of some honest party with probability $\frac{(2m-|I|)}{|\mathbb{F}|}\frac{|P|}{|\mathbb{F}|} = \frac{(2t+1)m(2m-|I|)}{|\mathbb{F}|^2}$. Now there can be at most $2m - |I|$ such e's (as there are $2m - |I|$ roots of $E(x)$). Any one of these e may enter into I of any honest party with probability $(2m - |I|)\frac{(2t+1)m(2m-|I|)}{|\mathbb{F}|^2}$. Putting the minimum value for I i.e $|I| = 0$ and value of $|\mathbb{F}|$, we get $\frac{(2t+1)m^3}{|\mathbb{F}|^2} \leq \frac{(2t+1)m^3\epsilon}{(\max(m^2,n))^2 n^6} \leq \frac{m^3\epsilon}{(\max(m^2,n))^2 n^5}$. Now irrespective of the relation between m and n, $\frac{m^3\epsilon}{(\max(m^2,n^2))^2 n^3} \ll \epsilon$ will hold. $\square$

Finally we have the following theorem.

Theorem 6.33 *In our statistical MPSI protocol (with $n = 3t+1$) every party learns the intersection set $S_1 \cap S_2 \cap \ldots \cap S_n$, except with probability at most ϵ. That is our MPSI has ϵ error in* **Correctness**. *Moreover, $\mathcal{A}_t$ will not get any extra information, other than what can be inferred from the data sets of the corrupted parties and the intersection of the data sets of all the parties. The protocol achieves the following bounds:*

1. **Round Complexity:** *Thirty seven rounds.*
2. **Communication Complexity:** *Private and broadcast communication of $\mathcal{O}((m^2n^3 + n^4)\log\frac{1}{\epsilon})$ bits.*

Proof: **Correctness:** Following the correctness of the four constituent phases, it is implied that the polynomial $F(x)$ will be reconstructed correctly, except with probability ϵ. Now by Lemma 6.32, every honest party learns correct

$I = S_1 \cap \ldots \cap S_n$ from $F(x)$, except with probability at most ϵ.

Secrecy: Secrecy is asserted as follows: From the secrecy of the fours constituent phases, it is easy to see that none of the intermediate sharing will be known to $\mathcal{A}_t$. Only $F(x)$ will be disclosed and thus known to $\mathcal{A}_t$. However, from Lemma 6.29, $F(x)$ does not leak any extra information to $\mathcal{A}_t$, other than what can be inferred by the data sets of the corrupted parties and the intersection of the data sets of all the parties.

The round complexity and communication complexity follows from the round and communication complexity of Input Phase, Preparation Phase, Computation Phase and Output Phase. □

6.9 Statistical MPSI Protocol with Optimal Resilience

In this section, we present a statistical MPSI protocol with $n = 2t + 1$ parties (i.e with optimal resilience) using the ideas presented for our statistical MPC protocol in Chapter 5. Our MPSI protocol takes $\Theta(1)$ rounds, privately communicates and broadcasts $\mathcal{O}((m^2n^4 + n^5)\log\frac{1}{\epsilon})$ bits. So our MPSI protocol achieves optimal resilience at the cost of a little bit higher communication complexity, in comparison to the MPSI protocol presented in the previous section.

As in [116, 129], our MPSI protocol tries to securely evaluate the function given in (6.1) and is divided into following four phases: Preparation Phase, Input Phase, Computation and Output Phase. The error probability of the overall protocol is ϵ. To bound the error probability by ϵ, all the computations in our protocol are performed over a finite field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^4 2^{-\kappa} \cdot \max(n, m^2)$. The relationship between κ and ϵ is derived from the relationship between ϵ and κ in our MPC protocol presented in Chapter 5, by putting values of c_M and c_O (the value for c_M and c_O are derived in the sequel). We assume that $n = \text{poly}(m)$. Any field element from field $\mathbb{F}$ can be represented by $\kappa = \log|\mathbb{F}| = \mathcal{O}(\log\frac{1}{\epsilon})$ bits (this can be derived using $n = \mathcal{O}(\log\frac{1}{\epsilon})$ and $n = \text{poly}(m)$).

In order to bound the error probability of our MPSI protocol by some specific values of ϵ, we find out the value of κ that satisfies $\epsilon \geq n^4 2^{-\kappa} \cdot \max(n, m^2)$. This value for κ will consequently determine the field $\mathbb{F}$ over which our protocol should work.

6.9.1 Preparation Phase

Let for $i = 1, \ldots, n$ polynomial $r^i(x)$ be expressed as $r^i(x) = b^{(0,i)} + b^{(1,i)}x + \ldots + b^{(m,i)}x^m$. Each of the random coefficients of $r^i(x)$ polynomials can be interpreted as a random gate. So there are $c_R = n(m+1)$ random gates (n polynomials $r^1(x), \ldots, r^n(x)$ have in total $n(m+1)$ random coefficients). Also there are $c_M = n(m+1)^2$ multiplication gates (computing $r^i(x)f^i(x)$ requires $(m+1)^2$ coefficient multiplications). There will be at most $(n(m+1)^2)$ additions of two values to compute $F(x)$. Hence $c_A = n(m+1)^2$. Finally as $F(x)$ is a $2m$ degree polynomial, it has $2m+1$ coefficients which need to be reconstructed/outputted. Hence $c_O = 2m+1$. So in preparation phase we will generate $2d^\star$-sharing of $c_R + c_M = n(m+1) + n(m+1)^2$ random multiplication triples, each having $\frac{\epsilon}{(2c_M + c_O)} \approx \frac{\epsilon}{n(m+1)^2}$

error, following the protocol for preparation phase (called as $\mathsf{PreparationPhase}$) of our statistical MPC protocol (presented in Chapter 5). Now consider the first c_R triples generated in preparation phase. The first component of these triples can be directly interpreted as $\langle\langle b^{(0,i)} \rangle\rangle_t, \ldots, \langle\langle b^{(m,i)} \rangle\rangle_t$ for $i = 1, \ldots, n$.

Theorem 6.34 *Except with error probability ϵ, the protocol for Preparation Phase produces correct $2d^\star$-sharing of $n(m+1)+n(m+1)^2$ secret multiplication triples, each having $\frac{\epsilon}{n(m+1)^2}$ error. The protocol has*

1. **Round Complexity:** *Twenty nine rounds.*
2. **Communication Complexity:** *Private and Broadcast communication of $\mathcal{O}((n^4m^2 + n^5)\log\frac{1}{\epsilon})$ bits.*

PROOF: The proof follows from Lemma 5.11 and 5.12 by substituting values of c_M, c_R, c_A and c_O as specified above. □

6.9.2 Input Phase

In the Input phase, every party $P_i \in \mathcal{P}$ represents his set $S_i = \{e_i^1, \ldots, e_i^m\}$ by a polynomial $f^i(x)$ of degree m where $f^i(x) = (x - e_i^1)\ldots(x - e_i^m) = a^{0i} + a^{1i}x + \ldots + a^{mi}x^m$. Since $a^{mi} = 1$ always, every party in $\mathcal{P}$ assumes 1 to be public on behalf of a^{mi}, for $i = 1, \ldots, n$ (see Remark 6.24). Now for $i = 1, \ldots, n$ and $j = 0, \ldots, m-1$, the parties generate $\langle\langle a^{ji} \rangle\rangle_t$ having $\frac{\epsilon}{\max(n^2, n(m+1)^2)}$ error, by executing the protocol for Input phase (i.e. $\mathsf{InputPhase}$) of our statistical MPC protocol (presented in Chapter 5), with $c_I = nm$.

Theorem 6.35 *Except with error probability ϵ, the protocol for* $\mathsf{Input\ Phase}$ *allows party P_i to generate $2d^\star$-sharings of all the coefficients of its polynomial $f^i(x)$, where each sharing will have $\frac{\epsilon}{\max(n^2, n(m+1)^2)}$ error. The protocol has:*

1. **Round Complexity:** *Five rounds.*
2. **Communication Complexity:** *Private and broadcast communication of $\mathcal{O}((n^4m + n^5)\log\frac{1}{\epsilon})$ bits.*

PROOF: The proof follows from Lemma 5.13 and 5.14 by substituting $c_I = nm$. □

6.9.3 Computation and Output Phase

After Preparation and Input phase, the parties jointly compute the coefficients of the polynomial $F(x) = \sum_{i=1}^{n} r^i f^i(x)$ in a shared manner. And finally the coefficients of $F(x)$ are reconstructed by each party. In the Output phase, each party locally evaluates $F(x)$ at each element of his private set. All the elements at which $F(x) = 0$ belongs to the intersection of the n sets with very high probability. The protocol for Computation and Output phase is given in Fig. 6.10.

Theorem 6.36 *Except with error probability ϵ, our protocol for Computation and Output phase computes intersection of the sets of individual parties with*

1. **Round Complexity:** *Four rounds.*

Figure 6.10: Computation Phase and Output phase of our Statistical MPSI Protocol

Computation Phase

1. All the parties in $\mathcal{P}$ compute $F^i(x) = r^i(x)f^i(x)$ such that $F^i(x) = c^{(0,i)} + c^{(1,i)}x + \ldots + c^{(2m,i)}x^{2m}$ is a $2m$ degree polynomial and all its coefficients $c^{(0,i)}, \ldots, c^{(2m,i)}$ are correctly $2d^\star$-shared. For $i = 1, \ldots, n$, the coefficients of all $F^i(x)$ are computed in parallel.

   (a) All parties in $\mathcal{P}$ compute in parallel:
   - $\langle\langle a^{(0,i)}b^{(0,i)}\rangle\rangle_t = \langle\langle a^{(0,i)}\rangle\rangle_t \langle\langle b^{(0,i)}\rangle\rangle_t$,
   - $\langle\langle a^{(0,i)}b^{(1,i)}\rangle\rangle_t = \langle\langle a^{(0,i)}\rangle\rangle_t \langle\langle b^{(1,i)}\rangle\rangle_t$,
   - $\ldots$,
   - $\langle\langle a^{(m,i)}b^{(m,i)}\rangle\rangle_t = \langle\langle a^{(m,i)}\rangle\rangle_t \langle\langle b^{(m,i)}\rangle\rangle_t$.

   For computing these products, we use the Beaver's circuit randomization technique, as used in our MPC protocol.

   (b) All parties in $\mathcal{P}$ compute
   - $\langle\langle c^{(0,i)}\rangle\rangle_t = \langle\langle a^{(0,i)}b^{(0,i)}\rangle\rangle_t$,
   - $\langle\langle c^{(1,i)}\rangle\rangle_t = \langle\langle a^{(0,i)}b^{(1,i)}\rangle\rangle_t + \langle\langle a^{(1,i)}b^{(0,i)}\rangle\rangle_t$,
   - $\langle\langle c^{(2,i)}\rangle\rangle_t = \langle\langle a^{(0,i)}b^{(2,i)}\rangle\rangle_t + \langle\langle a^{(2,i)}b^{(0,i)}\rangle\rangle_t + \langle\langle a^{(1,i)}b^{(1,i)}\rangle\rangle_t$,
   - $\ldots$,
   - $\langle\langle c^{(2m,i)}\rangle\rangle_t = \langle\langle a^{(m,i)}b^{(m,i)}\rangle\rangle_t$.

2. All the parties in $\mathcal{P}$ compute $F(x) = \sum_{i=1}^{n} F^i(x)$ such that $F(x) = d^0 + d^1x + \ldots + d^{2m}x^{2m}$ is a $2m$ degree polynomial and all its coefficients $d^0, \ldots, d^{2m}$ are correctly $2d^\star$-shared. For this all parties in $\mathcal{P}$ compute $\langle\langle d^j\rangle\rangle_t = \sum_{i=1}^{n}\langle\langle c^{ji}\rangle\rangle_t$ for $j = 0, \ldots, 2m$.

Output Phase

1. The coefficients $d^0, \ldots, d^{2m}$ of $F(x)$ are privately reconstructed by each party. For that parties invoke protocol 5VSS-Rec (of Chapter 4), as in our statistical MPC protocol presented in Chapter 5.

2. Each P_i with his private set $S_i = \{e_i^1, \ldots, e_i^m\}$ locally checks whether $F(e_i^k) \stackrel{?}{=} 0$ for $k = 1, \ldots, m$. If $F(e_i^k) = 0$, the P_i adds e_i^k in a set I_i (initially $I_i = \emptyset$). P_i outputs I_i as the intersection set $S_1 \cap S_2 \ldots, \cap S_n$.

2. **Communication Complexity:** *Private and broadcast communication of* $\mathcal{O}(n^4m^2 \log\frac{1}{\epsilon})$ *bits.*

PROOF: Follows from Lemma 5.15 and 5.16 by substituting $c_M = \mathcal{O}(nm^2)$, $c_O = 2m + 1$ and $\mathcal{D} = 1$. □

6.9.4 Our New MPSI with Optimal Resilience

Now our protocol for statistical MPSI with $n = 2t + 1$ is: (a) Invoke protocol for Preparation phase and Input phase in parallel; (c) Invoke protocol for

Computation and Output phase.

Theorem 6.37 *In our statistical MPSI protocol (with $n = 2t + 1$) every party learns the intersection set $S_1 \cap S_2 \cap \ldots \cap S_n$, except with probability at most ϵ. That is our MPSI has ϵ error in* **Correctness***. Moreover, $\mathcal{A}_t$ will not get any extra information, other than what can be inferred by the data sets of the corrupted parties and the intersection of the data sets of all the parties. The protocol achieves the following bounds:*

1. **Round Complexity:** *Thirty three rounds.*
2. **Communication Complexity:** *Private and broadcast communication of $\mathcal{O}((m^2n^4 + n^5)\log\frac{1}{\epsilon})$ bits.*

PROOF: **Correctness** and **Secrecy** can be argued in the same way as done for our MPSI protocol with $n = 3t+1$ parties. The round complexity and communication complexity follow from the round and communication complexity of Input Phase, Preparation Phase, Computation Phase and Output Phase. □

We will conclude this section with a comparison of our optimally resilient MPSI with the MPSI protocols that may be derived from existing MPC protocols with optimal resilience by substituting the number of gates as done in subsection 6.3.1.

6.9.5 Our MPSI Protocol with $n = 2t + 1$ vs. Existing General MPC Protocols

Assume that an MPSI protocol computes the function given in (6.1), using general MPC protocol. The arithmetic circuit, representing the function in (6.1), will roughly require $c_M = n(m+1)^2$ multiplication gates. This is because computing $r^i(x)f^i(x)$ requires $(m+1)^2$ coefficient multiplications. And since all the multiplications can be evaluated in parallel, we have the multiplicative depth of the circuit as one i.e $\mathcal{D} = 1$.

Now in Table 6.3, we have summarized the communication complexity and round complexity of MPSI protocols that may be derived from existing MPC protocols with optimal resilience. This is done by putting $c_M = n(m+1)^2$ and $\mathcal{D} = 1$ in the communication and round complexity of existing MPC protocols with optimal resilience (i.e with $n = 2t + 1$).

From Table 6.3, we find that our protocol incurs much lesser communication complexity than the protocol of [138, 4, 6, 48, 49] while achieving a round complexity of same or less order. But the protocol of [12] provides slightly better communication complexity than ours at the cost of increased round complexity.

6.10 Conclusion and Open Problems

In this chapter, we have presented a detailed analysis of the round complexity and communication complexity of the statistical MPSI protocol of [116] and presented a new protocol with significant improvement over the same. Towards this, we have designed a new statistical VSS protocol and new sub-protocols like Upgrade1Dto2D. These protocols along with existing techniques from the literature, led to our efficient statistical MPSI protocol. We have also designed a statistical

Table 6.3: Comparison of our MPSI with the general MPC protocols that securely compute (6.1).

Reference	Communication Complexity in bits		Round Complexity
	Private	Broadcast	
[138]	$\Omega(m^2 n^6 (\log \frac{1}{\epsilon})^4)$	$\Omega(m^2 n^6 (\log \frac{1}{\epsilon})^4)$	$\mathcal{O}(1)$
[4, 6]	$\Omega(m^2 n^6 (\log \frac{1}{\epsilon})^4)$	$\Omega(m^2 n^6 (\log \frac{1}{\epsilon})^4)$	$\mathcal{O}(1)$
[48]	$\mathcal{O}(m^2 n^6 \log \frac{1}{\epsilon})$	$\mathcal{O}(m^2 n^6 \log \frac{1}{\epsilon})$	$\mathcal{O}(n)$
[49]	$\mathcal{O}(m^2 n^6 \log \frac{1}{\epsilon})$	$\mathcal{O}(m^2 n^6 \log \frac{1}{\epsilon})$	$\mathcal{O}(n)$
[12]	$\mathcal{O}(m^2 n^3 \log \frac{1}{\epsilon})$	$\mathcal{O}(n^3 \log \frac{1}{\epsilon})$	$\mathcal{O}(n^2)$
This chapter	$\mathcal{O}((m^2 n^4 + n^5) \log \frac{1}{\epsilon})$	$\mathcal{O}((m^2 n^4 + n^5) \log \frac{1}{\epsilon})$	33

MPSI protocol with optimal resilience. We now conclude this chapter with a few open questions:

Open Problem 9 *Can we improve the round complexity and communication complexity of information theoretically secure MPSI protocol?*

Open Problem 10 *Can we design a perfect MPSI protocol using a direct method (and without using circuit based approach of general MPC protocol)?*

The later question calls for a new way of solving MPSI problem that is different from the method followed in this thesis (based on finding common roots of n polynomials). This is because the current method has an inherent error probability involved, as mentioned in Subsection 6.1.2.

Part II

Results in Asynchronous Network

Chapter 7

Efficient Asynchronous Information Checking Protocols

In this chapter, we focus on Information Checking Protocol (ICP) in asynchronous network, called as AICP (asynchronous ICP). Recall that ICP is a tool for authenticating messages in the presence of *computationally unbounded* corrupted parties. Much like the ICP in synchronous network is instrumental in constructing statistical VSS and WSS protocols, AICP is a vital building block for designing statistical AVSS and AWSS protocols. Here we present two AICPs with slight variations in their properties and communication complexity. Both the protocols are highly efficient and they will be used for designing AVSS protocols with different properties (details will appear in the next chapter). At the end of this chapter, we present a discussion on our AICPs, comparing and contrasting their properties and motivations. We also compare the protocols with the only known existing AICP of [39].

7.1 Introduction

7.1.1 Existing Literature and Definition of Asynchronous ICP or AICP

The notion of ICP was first introduced by Rabin et al. [138] who have designed an ICP in *synchronous* settings. The ICP of Rabin et al. was also used as a tool by Canetti et al. [39] in asynchronous network for designing their Asynchronous BA (ABA) scheme.

Canetti et al. [39] have defined AICP as a protocol executed among three parties: a *dealer* D, an *intermediary* INT and a *verifier* R. The dealer D hands over a secret value s to INT. At a later stage, INT is required to hand over s to R and convince R that s is indeed the value which INT received from D. So the basic definition of ICP involves only a *single* verifier R.

7.1.2 New Definition, Model, Structure and Properties of AICP

Similar to the extension that we have done for ICP in Chapter 2, we first extend the basic definition of AICP so that it can deal with multiple verifiers and multiple secrets simultaneously. Specifically, our AICP can deal with n verifiers denoted by $\mathcal{P}$ and ℓ secrets concurrently. Moreover, we assume that dealer D and INT belong to $\mathcal{P}$. So now our AICP is executed among three entities: a dealer $D \in \mathcal{P}$, an intermediary $INT \in \mathcal{P}$ and the entire set $\mathcal{P}$ acting as verifiers. The dealer D

hands over a secret S to INT. At a later stage, INT is required to hand over S to the verifiers in $\mathcal{P}$ and convince the honest parties in $\mathcal{P}$ that S is indeed the secret which INT received from D. Extending the AICP for multiple verifiers and multiple secrets, will be later helpful in using AICP as a tool in our AVSS and AWSS protocols.

As mentioned earlier, our multiple secret, multiple receiver AICP is useful in the design of efficient protocols for statistical AVSS and AWSS. Statistical AVSS is possible iff $n \geq 3t + 1$ and for the design of statistical AVSS with optimal resilience, we work with $n = 3t + 1$. As our AICPs is useful in such context, we design our AICPs as well with $n = 3t + 1$. Now in this chapter, we use the following network model.

7.1.2.1 The Network and Adversary Model for AICP

We assume that there are n parties (in this chapter, we will also call them as verifiers), say $\mathcal{P} = \{P_1, \ldots, P_n\}$, where every two parties are directly connected by a secure channel and t out of the n parties can be under the influence of a *computationally unbounded Byzantine (active) adversary*, denoted as $\mathcal{A}_t$. Furthermore, we assume $n = 3t+1$. Also $D, INT \in \mathcal{P}$ are two specific parties where D is called as *Dealer* and INT is referred to as *Intermediary*. The adversary $\mathcal{A}_t$ may corrupt D as well as INT. The Byzantine adversary $\mathcal{A}_t$ completely dictates the parties under its control and can force them to deviate from a protocol, in any arbitrary manner. The adversary $\mathcal{A}_t$ is *static* and thus can corrupt some t parties before the start of the protocol. We assume $\mathcal{A}_t$ to be *rushing* [125, 91, 48], who may choose to first listen all the messages sent to the corrupted parties by the honest parties, before allowing the corrupted parties to send their messages. The parties not under the influence of $\mathcal{A}_t$ are called *honest or uncorrupted*.

The underlying network is asynchronous, where the communication channels between the parties have arbitrary, yet finite delay (i.e the messages are guaranteed to reach eventually). To model this, $\mathcal{A}_t$ is given the power to schedule the delivery of *all* messages in the network. However, $\mathcal{A}_t$ can only schedule the messages communicated between honest parties, without having any access to the contents of the message.

7.1.2.2 The Structure of AICP

As in [39, 35], our AICP is also structured into sequence of following three phases:

1. **Generation Phase**: This phase is initiated by D. Here D hands over the secret S containing ℓ elements from $\mathbb{F}$ to *intermediary* INT. In addition, D sends some *authentication information* to INT and some *verification information* to individual verifiers in $\mathcal{P}$.

2. **Verification Phase**: This phase is initiated by INT to acquire an IC Signature on S that will be later accepted by every honest verifiers in $\mathcal{P}$. Depending on the behavior of D (i.e whether honest or corrupted), INT may or may not receive IC signature from D. When INT receives IC signature, he decides to continue AICP and later participate in **Revelation Phase**. On the other hand, when INT does not receive IC signature, he aborts AICP and does not participate in **Revelation Phase** later. The IC signature (when INT receives it), denoted by $ICSig(D, INT, \mathcal{P}, S)$ is

either S along with the *authentication information* which is held by INT at the end of **Verification Phase** or only S.

3. **Revelation Phase**: This phase is carried out by INT (only when he receives $ICSig(D, INT, \mathcal{P}, S)$ from D by the end of **Verification Phase**) and the verifiers in $\mathcal{P}$. **Revelation Phase** can be presented in two flavors:

   (a) *Public Revelation* of $ICSig(D, INT, \mathcal{P}, S)$ to all the verifiers in $\mathcal{P}$: Here all the verifiers can publicly verify whether INT indeed received IC signature on S from D. If they are convinced then every verifier P_i sets $\mathsf{Reveal}_i = S$. Otherwise every P_i sets $\mathsf{Reveal}_i = NULL$.

   (b) P_α*-private-revelation* of $ICSig(D, INT, \mathcal{P}, S)$: Here INT privately reveals $ICSig(D, INT, \mathcal{P}, S)$ to *only* P_α, where $P_\alpha \in \mathcal{P}$. After doing some checking, if P_α believes that INT indeed received IC signature on S from D then P_α sets $\mathsf{Reveal}_\alpha = S$. Otherwise P_α sets $\mathsf{Reveal}_\alpha = NULL$.

In our ICP in synchronous network, we have mentioned and implemented only *Public Revelation* of $ICSig(D, INT, \mathcal{P}, S)$. In all the applications of our ICP in synchronous network (as shown in this thesis), we required only the *Public Revelation*. But in asynchronous network, while we will require *Public Revelation* in some instances, we will also require P_α*-private-revelation* in some other instances (as will be demonstrated in the subsequent chapters).

7.1.2.3 The Properties of AICP

Any AICP should satisfy the following properties, assuming public revelation of signature (these properties are almost same as the properties of AICP defined in [39]). In the properties, ϵ denotes the error probability of AICP.

1. **AICP-Correctness1:** If D and INT are *honest*, then $ICSig(D, INT, \mathcal{P}, S)$ will be accepted in **Revelation Phase** by each *honest* verifier.

2. **AICP-Correctness2:** If an *honest* INT holds an $ICSig(D, INT, \mathcal{P}, S)$ at the end of **Verification Phase**, then $ICSig(D, INT, \mathcal{P}, S)$ will be accepted in **Revelation Phase** by each honest verifier, except with probability ϵ.

3. **AICP-Correctness3:** If D is *honest*, then during **Revelation Phase**, with probability at least $(1 - \epsilon)$, every $ICSig(D, INT, \mathcal{P}, S')$ with $S' \neq S$ produced by a *corrupted* INT will not be accepted by an *honest* verifier.

4. **AICP-Secrecy:** If D and INT are *honest* and INT has not started **Revelation Phase**, then $\mathcal{A}_t$ will have no information about S.

For AICP with P_α*-private-revelation* in **Revelation Phase**, the above properties can be modified by replacing "every/any honest verifier" with "honest P_α".

Both of our AICPs are presented with *Public Revelation* as well as P_α*-private-revelation*; but later depending on our requirement (described in detail in subsequent sections) we use one of the AICPs with *Public Revelation*, while other AICP with P_α*-private-revelation*. For our protocols, we need a basic tool called A-cast that allows any party in $\mathcal{P}$ to send some information identically to all the parties in $\mathcal{P}$.

7.2 A-cast: Asynchronous Broadcast

A-cast is an asynchronous broadcast primitive. It was introduced and elegantly implemented by Bracha [29] with $n = 3t + 1$ parties.

Definition 7.1 (A-cast [35]) *: Let Π be a protocol executed among the set of parties $\mathcal{P}$ and initiated by a special party caller sender $S \in \mathcal{P}$, having input m (the message to be sent). Π is an A-cast protocol tolerating $\mathcal{A}_t$ if the following hold, for every behavior of $\mathcal{A}_t$ and every input m:*

1. **Termination:**
   (a) *If S is honest, then all honest parties in $\mathcal{P}$ will eventually terminate Π;*
   (b) *If any honest party terminates Π, then irrespective of the nature of S all honest parties will eventually terminate Π.*
2. **Correctness:**
   (a) *If the honest parties terminate Π, then they do so with a common output m^*;*
   (b) *Furthermore, if the sender S is honest then $m^* = m$.*

For the sake of completeness, we recall Bracha's A-cast protocol from [35] and present it in Fig. 7.1. For convenience, we denote the protocol of [29] as Bracha-A-cast$(S, \mathcal{P}, M)$, where M is the message that the sender S wants to send and $|M| \geq 1$ (in bits).

Figure 7.1: Bracha's A-cast Protocol with $n = 3t + 1$

Bracha-A-cast$(S, \mathcal{P}, M)$

Code for the sender S (with input M): only S executes this code

1. Send message (MSG, M) privately to all the parties.

Code for party P_i: every party in $\mathcal{P}$ executes this code

1. Upon receiving a message (MSG, M), send $(ECHO, M)$ privately to all parties.
2. Upon receiving $n - t$ messages $(ECHO, M')$ that agree on the value of M', send $(READY, M')$ privately to all the parties.
3. Upon receiving $t + 1$ messages $(READY, M')$ that agree on the value of M', send $(READY, M')$ privately to all the parties.
4. Upon receiving $n - t$ messages $(READY, M')$ that agree on the value of M', send (OK, M') privately to all the parties, accept M' as the output message and terminate the protocol.

Theorem 7.2 ([35]) *Protocol* Bracha-A-cast *privately communicates* $\mathcal{O}(|M|n^2)$ *bits to* A-cast *a message* M *of size* $|M|$ *bits.*

Notation 7.3 (Convention for Using Bracha's A-cast Protocol) *In the rest of the thesis, we use the following convention: By saying that '*P_i A-casts M*', we mean that* P_i *as a sender, initiates* Bracha-A-cast$(P_i, \mathcal{P}, M)$*. Then by saying that '*P_j *receives* M *from the* A-cast *of* P_i*', we mean that* P_j *terminates the execution of* Bracha-A-cast$(P_i, \mathcal{P}, M)$*, with* M *as the output.*

7.3 Our First AICP

In the following, we present an informal idea of our novel AICP called MVMS-AICP-I and subsequently describe protocol MVMS-AICP-I in Fig. 7.2 and 7.3. The protocol has a error probability of ϵ. To bound the error probability by ϵ, our protocol works over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n\kappa 2^{-\kappa}$ and the value of ϵ. Specifically the the *minimum* value of κ that satisfies $\epsilon \geq n\kappa 2^{-\kappa}$ will determine the field $\mathbb{F}$. The relation between ϵ and κ implies that we have $|\mathbb{F}| \geq \frac{n\kappa}{\epsilon}$. Now each element from the field is represented by $\kappa = \log|\mathbb{F}| = \mathcal{O}(\log\frac{1}{\epsilon})$ bits (this can be derived using $n = \mathcal{O}(\log\frac{1}{\epsilon})$).

The Intuition: D selects a random polynomial $f(x)$ of degree $\ell + t\kappa$, whose first ℓ coefficients are the elements of S and delivers $f(x)$ to INT. In addition, D privately gives the value of $f(x)$ at κ random *evaluation points* to each individual verifier. This distribution of information by D helps to achieve **AICP-Correctness3** property. Specifically, if D is *honest*, then a *corrupted* INT cannot produce an *incorrect* $f'(x) \neq f(x)$ during **Revelation Phase** without being detected by an *honest* verifier. This is because a corrupted INT will have no information about the evaluation points of an honest verifier and hence with very high probability, $f'(x)$ will not match with the evaluation points held by an honest verifier.

The above distribution of information by D also maintains **AICP-Secrecy** property. This is because the degree of $f(x)$ is $\ell + t\kappa$ and $\mathcal{A}_t$ will know the value of $f(x)$ at most at $t\kappa$ evaluation points.

However, a *corrupted* D might do the following: he may distribute $f(x)$ to INT and value of some other polynomial (different from $f(x)$) to each honest verifier. To avoid this situation, INT and the verifiers interact in *zero knowledge* fashion, using *cut-and-choose* technique to check the consistency of $f(x)$ and the values of $f(x)$ held by individual verifier. The specific details of the *cut-and-choose*, along with other formal steps of protocol MVMS-AICP-I are given in Fig. 7.2 and 7.3.

We now prove the properties of protocol MVMS-AICP-I considering the P_α-private-revelation of $ICSig(D, INT, \mathcal{P}, S)$ (later MVMS-AICP-I will be used with its P_α-private-revelation only). The proofs can be twisted little bit to obtain the proofs for public revelations.

Lemma 7.4 (AICP-Correctness1) *If* D*,* INT *and* P_α *are honest, then* S *will be accepted by* P_α*.*

Proof: If D is honest then he will honestly deliver $f(x)$ to INT and its values at κ points to individual verifier. So eventually, the condition stated in step 2(a)

Figure 7.2: Our First AICP with $n = 3t + 1$ Verifiers.

Protocol MVMS-AICP-I$(D, INT, \mathcal{P}, S, \epsilon)$

Generation Phase: Gen$(D, INT, \mathcal{P}, S, \epsilon)$

1. D selects a random $\ell + t\kappa$ degree polynomial $f(x)$ whose lower order ℓ coefficients are the secrets in $S = (s^1, \ldots, s^\ell)$. D also picks $n\kappa$ random, non-zero, distinct *evaluation points* from $\mathbb{F}$, denoted by $\alpha_1^i, \ldots, \alpha_\kappa^i$, for $i = 1, \ldots, n$.
2. D privately sends $f(x)$ to INT and the verification tags $z_1^i = (\alpha_1^i, a_1^i), \ldots, z_\kappa^i = (\alpha_\kappa^i, a_\kappa^i)$ to party P_i. Here $a_j^i = f(\alpha_j^i)$, for $j = 1, \ldots, \kappa$.

Verification Phase: Ver$(D, INT, \mathcal{P}, S, \epsilon)$

1. Every verifier P_i randomly partitions the index set $\{1, \ldots, \kappa\}$ into two sets I^i and $\overline{I^i}$ of equal size and sends I^i and z_j^i for all $j \in I^i$ to INT.
2. Local Computation (only for INT):
   (a) For every verifier P_i from which INT has received I^i and corresponding verification tags, INT checks whether for *every* $j \in I^i$, $f(\alpha_j^i) \stackrel{?}{=} a_j^i$.
   (b) If for at least $2t+1$ verifiers, the above condition is satisfied, then INT sets $ICSig(D, INT, \mathcal{P}, S) = f(x)$ and concludes that he has received $ICSig(D, INT, \mathcal{P}, S)$ from D.
   (c) If for at least $t + 1$ verifiers, the above condition is not satisfied, then INT sets $ICSig(D, INT, \mathcal{P}, S) = NULL$ and concludes that he has not received $ICSig(D, INT, \mathcal{P}, S)$ from D.

of **Verification Phase** will be satisfied for at least $2t + 1$ verifiers and hence INT, who is honest in this case will set $ICSig(D, INT, \mathcal{P}, S) = f(x)$. Now it is easy to see that the condition stated in step 3(a) of protocol Reveal-Private will be eventually satisfied, corresponding to the honest verifiers in $\mathcal{P}$ (there are at least $2t+1$ honest verifiers). Hence P_α, who is honest in this case, will eventually accept $ICSig(D, INT, \mathcal{P}, S)$ at the end of Reveal-Private. □

Lemma 7.5 (AICP-Correctness2) *If an honest INT holds an $ICSig(D, INT, \mathcal{P}, S)$ at the end of* **Verification Phase***, then $ICSig(D, INT, \mathcal{P}, S)$ will be accepted in Reveal-Private by honest P_α, except with probability ϵ.*

Proof: We have to consider the case when D *is corrupted* as otherwise the proof will follow from Lemma 7.4. Since INT is honest and it holds an $ICSig(D, INT, \mathcal{P}, S)$ at the end of **Verification phase**, INT has ensured that for at least $2t + 1$ verifiers the condition specified in step 2(a) of **Verification phase** has been satisfied. Let $\mathcal{H}$ be the set of *honest* verifiers among these $2t + 1$ verifiers. Note that $|\mathcal{H}| \geq t + 1$. To prove the lemma, we prove that corresponding to each verifier in $\mathcal{H}$, the condition stated in step 3(a) of Reveal-Private will be satisfied with very high probability. Note that corresponding to a verifier P_i in $\mathcal{H}$, the condition stated in step 3(a) of Reveal-Private will fail if for *all* $j \in \overline{I^i}$, $f(\alpha_j^i) \neq a_j^i$. This implies that (corrupted) D must have distributed $f(x)$ (to

Figure 7.3: Our First AICP with $n = 3t + 1$ Verifiers.

Protocol MVMS-AICP-I$(D, INT, \mathcal{P}, S, \epsilon)$

Revelation Phase:

Reveal-Private$(D, INT, \mathcal{P}, S, P_\alpha, \epsilon)$**:** P_α*-private-revelation* of $ICSig(D, INT, \mathcal{P}, S)$

1. To party P_α, INT sends $ICSig(D, INT, \mathcal{P}, S) = f(x)$.
2. To party P_α, every verifier P_i sends the index set $\overline{I^i}$ and all z^i_j such that $j \in \overline{I^i}$.
3. Local Computation (only for P_α):
   (a) Upon receiving $f(x)$ from INT and the values from verifier P_i, check whether for *some* $j \in \overline{I^i}$, $f(\alpha^i_j) \stackrel{?}{=} a^i_j$.
   (b) If for at least $t + 1$ verifiers the above condition is satisfied, then accept $ICSig(D, INT, \mathcal{P}, S)$ and set $\mathsf{Reveal}_\alpha = S$, where S is the lower order ℓ coefficients of $f(x)$.
   (c) If for at least $2t+1$ verifiers the above condition is not satisfied, then reject $ICSig(D, INT, \mathcal{P}, S)$ and set $\mathsf{Reveal}_\alpha = NULL$.

Reveal-Public$(D, INT, \mathcal{P}, S, \epsilon)$**:** *Public Revelation* of $ICSig(D, INT, \mathcal{P}, S)$

1. INT A-casts $ICSig(D, INT, \mathcal{P}, S) = f(x)$.
2. Code for P_i: (Every party in $\mathcal{P}$ executes this code.)
   (a) Upon receiving $f(x)$ from INT, check whether for *some* $j \in \overline{I^i}$, $f(\alpha^i_j) \stackrel{?}{=} a^i_j$. If yes, A-cast `Accept`. Else A-cast `Reject`.
   (b) If `Accept` is received from the A-cast of at least $t + 1$ verifiers, then accept $ICSig(D, INT, \mathcal{P}, S)$ and set $\mathsf{Reveal}_i = S$, where S is the lower order ℓ coefficients of $f(x)$.
   (c) If `Reject` is received from the A-cast of at least $2t+1$ verifiers, then reject $ICSig(D, INT, \mathcal{P}, S)$ and set $\mathsf{Reveal}_i = NULL$.

INT) and z^i_j (to P_i) *inconsistently* for *all* $j \in \overline{I^i}$ and it so happens that P_i has partitioned $\{1, \ldots, \kappa\}$ into I^i and $\overline{I^i}$ during **Verification Phase**, such that $\overline{I_i}$ contains only inconsistent tuples (z^i_j's). Thus corresponding to a verifier $P_i \in \mathcal{H}$, the probability that the condition stated in step 3(a) of Reveal-Private fails is same as the probability of P_i selecting all consistent (inconsistent) tuples in I^i ($\overline{I^i}$), which is $\frac{1}{\binom{\kappa}{\kappa/2}} < \frac{1}{2^\kappa} \leq \frac{\epsilon}{n\kappa}$. Now as there are at least $t + 1$ parties in $\mathcal{H}$, except with probability $(t+1)\frac{\epsilon}{n\kappa} \approx \frac{\epsilon}{\kappa} < \epsilon$, P_α will eventually find step 3(a) of Reveal-Private to be true for all parties in $\mathcal{H}$ and will accept $ICSig(D, INT, \mathcal{P}, S)$. □

Lemma 7.6 (AICP-Correctness3) *If D is honest, then during Reveal-Private, with probability at least $(1-\epsilon)$, every $ICSig(D, INT, \mathcal{P}, S')$ with $S' \neq S$ produced by a corrupted INT will be rejected by honest verifier P_α.*

Proof: It is easy to see that $S' \neq S$ produced by a *corrupted* INT will be accepted by an *honest* P_α, if the condition stated in step 3(a) of Reveal-Private gets satisfied corresponding to *at least one honest* verifier (for t corrupted verifiers, the condition may always satisfy). However, the condition will be satisfied corresponding to honest verifier P_i if corrupted INT can *correctly guess* a verification tag z_i^j for at least one $j \in \overline{I^i}$, which he can do with probability $\frac{\frac{\kappa}{2}}{|\mathbb{F}|} \leq \frac{\epsilon}{n}$. Now since there are at least $2t+1$ honest verifiers, the probability that INT can guess the above for some honest verifier is at most $(2t+1)\frac{\epsilon}{n} \approx \epsilon$. This implies that with probability $(1-\epsilon)$, for all the honest verifiers the condition stated in step 3(a) of Reveal-Private will *not* be satisfied. Thus, with probability at least $(1-\epsilon)$, every $ICSig(D, INT, \mathcal{P}, S')$ with $S' \neq S$ produced by a *corrupted* INT will be rejected by *honest* verifier P_α. □

Lemma 7.7 (AICP-Secrecy) *If D and INT are honest and INT has not started Reveal-Private, then S is information theoretically secure from $\mathcal{A}_t$.*

PROOF: If D and INT are honest, then at the end of **Verification Phase**, $\mathcal{A}_t$ will get $t\kappa$ distinct values on $f(x)$. However, $f(x)$ is of degree $\ell + t\kappa$ and hence the lower order ℓ coefficients of $f(x)$ which are the elements of S will remain information theoretically secure. □

Lemma 7.8 (Communication Complexity of MVMS-AICP-I)

- *Protocol Gen privately communicates $\mathcal{O}((\ell + n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits.*
- *Protocol Ver privately communicates $\mathcal{O}((n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits.*
- *Protocol Reveal-Private privately communicates $\mathcal{O}((\ell + n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits.*
- *Protocol Reveal-Public A-casts $\mathcal{O}((\ell + n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits.*

PROOF: In protocol Gen, D privately gives $\ell+t\kappa$ field elements to INT and κ field elements to each verifier. Since each field element can be represented by κ bits and $\kappa = \mathcal{O}(\log\frac{1}{\epsilon})$, protocol Gen incurs a private communication of $\mathcal{O}((\ell+n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits. In protocol Ver, every verifier privately sends $\frac{\kappa}{2}$ field elements to INT, thus incurring a total private communication of $\mathcal{O}((n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits. In protocol Reveal-Private, INT sends to P_α the polynomial $f(x)$, consisting of $\ell + t\kappa$ field elements, while each verifier sends $\overline{I^i}$ and corresponding verification tags. So Reveal-Private involves private communication of $\mathcal{O}((\ell + n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits. In Reveal-Public, INT A-casts $f(x)$ and the verifiers A-cast either `Accept` or `Reject`. So Reveal-Public require A-cast communication of $\mathcal{O}((\ell + n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits. □

Theorem 7.9 *Protocol MVMS-AICP-I is an efficient AICP.*

PROOF: The theorem follows from Lemma 7.4, Lemma 7.5, Lemma 7.6 and Lemma 7.7. □

In the sequel, we present our second AICP.

7.4 Our Second AICP

We now present our second AICP called MVMS-AICP-II. The idea of MVMS-AICP-II is very similar to the ICP presented in Chapter 2. Hence we directly present the protocol in Fig. 7.4 and 7.5.

To bound the error probability by ϵ, our protocol MVMS-AICP-II operates over field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n2^{-\kappa}$. Hence we have $|\mathbb{F}| \geq \frac{n}{\epsilon}$ and each element from the field is represented by $\kappa = \log|\mathbb{F}| = \mathcal{O}(\log\frac{1}{\epsilon})$ bits (this can be derived using $n = \mathcal{O}(\log\frac{1}{\epsilon})$).

Figure 7.4: Our second AICP with $n = 3t + 1$

Protocol MVMS-AICP-II$(D, INT, \mathcal{P}, S, \epsilon)$

Generation Phase: Gen$(D, INT, \mathcal{P}, S, \epsilon)$

1. D picks and sends the following to INT:
   (a) A random degree-$(\ell + t)$ polynomial $F(x)$ over $\mathbb{F}$, such that the lower order ℓ coefficients of $F(x)$ are elements of S.
   (b) A random degree-$(\ell + t)$ polynomial $R(x)$ over $\mathbb{F}$.
2. D privately sends the following to every verifier P_i:
   (a) (α_i, v_i, r_i), where $\alpha_i \in \mathbb{F} - \{0\}$ is random (all α_i's are distinct).
   (b) $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$.

   The polynomials $F(x), R(x)$ is called **authentication information**, while for $i = 1, \ldots, n$, the values (α_i, v_i, r_i) are called **verification information**.

Verification Phase: Ver$(D, INT, \mathcal{P}, S, \epsilon)$

1. For $i = 1, \ldots, n$, verifier P_i sends a `Received-From-D` signal to INT after receiving (α_i, v_i, r_i) from D.
2. Upon receiving `Received-From-D` from $2t + 1$ verifiers, INT creates a set $ReceivedSet = \{P_i \mid INT \text{ received } \texttt{Received-From-D} \text{ signal from } P_i\}$ (clearly $|ReceivedSet| = 2t + 1$). INT then chooses a random $d \in \mathbb{F} \setminus \{0\}$ and A-casts $(d, B(x), ReceivedSet)$, where $B(x) = dF(x) + R(x)$.
3. D checks $dv_i + r_i \stackrel{?}{=} B(\alpha_i)$ for every $P_i \in ReceivedSet$. If D finds any inconsistency, he A-casts S. Otherwise D A-casts `OK`.
4. Upon receiving the A-cast of D, INT sets
   (a) $ICSig(D, INT, \mathcal{P}, S) = F(x)$ if `OK` is received from the A-cast of D.
   (b) $ICSig(D, INT, \mathcal{P}, S) = S$ if S is received from the A-cast of D.

From Fig. 7.4 and 7.5, it is clear that the idea of MVMS-AICP-II is very similar to the ICP presented in Chapter 2. Hence the proofs for MVMS-AICP-II go in the line of the proofs of the ICP in Chapter 2. For the sake of completeness, we present all the proofs of MVMS-AICP-II in the sequel considering its public revelation (later MVMS-AICP-II will be used with its public revelation only). The proofs can be twisted little bit to obtain the proofs for P_α-private-revelation.

Figure 7.5: Our second AICP with $n = 3t + 1$

Protocol MVMS-AICP-II$(D, INT, \mathcal{P}, S, \epsilon)$

Revelation Phase:

Reveal-Public$(D, INT, \mathcal{P}, S, \epsilon)$: Public Revelation of $ICSig(D, INT, \mathcal{P}, S)$

1. INT A-casts $ICSig(D, INT, \mathcal{P}, S)$.
2. On receiving $ICSig(D, INT, \mathcal{P}, S)$ from the A-cast of INT, verifier $P_i \in ReceivedSet$ who indeed sent Received-From-D to INT during Ver, A-casts `Accept` in the following conditions.
   (a) If $ICSig(D, INT, \mathcal{P}, S) = S$ and the S A-casted by D during Ver is same as $ICSig(D, INT, \mathcal{P}, S)$.
   (b) If $ICSig(D, INT, \mathcal{P}, S) = F(x)$ and one of the following holds.
      i. **C1:** $v_i = F(\alpha_i)$; OR
      ii. **C2:** $B(\alpha_i) \neq dv_i + r_i$, where $B(x)$ was A-casted by INT during Ver.

   Otherwise, P_i A-casts `Reject`.

Local Computation (By Every Verifier in $\mathcal{P}$): If at least $(t + 1)$ verifiers from $ReceivedSet$ have A-casted `Accept` during Reveal-Public then accept $ICSig(D, INT, \mathcal{P}, S)$ and set $\text{Reveal}_i = S$. Else reject $ICSig(D, INT, \mathcal{P}, S)$ and set $\text{Reveal}_i = NULL$.

Reveal-Private$(D, INT, \mathcal{P}, P_\alpha, S, \epsilon)$:P_α-private-revelation of $ICSig(D, INT, \mathcal{P}, S)$

1. INT sends $ICSig(D, INT, \mathcal{P}, S)$ to P_α.
2. Verifier P_i sends (α_i, v_i, r_i) to P_α.
3. Local Computation by P_α.
   (a) Upon receiving $ICSig(D, INT, \mathcal{P}, S)$ from INT and the values from verifier P_i, accept P_i if
      i. If $ICSig(D, INT, \mathcal{P}, S) = S$ and the S A-casted by D during Ver is same as $ICSig(D, INT, \mathcal{P}, S)$.
      ii. If $ICSig(D, INT, \mathcal{P}, S) = F(x)$ and one of the following holds.
         A. **C1:** $v_i = F(\alpha_i)$; OR
         B. **C2:** $B(\alpha_i) \neq dv_i + r_i$, where $B(x)$ was A-casted by INT during Ver.

      Otherwise reject P_i.
   (b) If at least $t + 1$ verifiers from $ReceivedSet$ are accepted, then accept $ICSig(D, INT, \mathcal{P}, S)$ and set $\text{Reveal}_\alpha = S$, where S is lower order ℓ coefficients of $f(x)$.
   (c) If at least $t + 1$ verifiers from $ReceivedSet$ are rejected, then reject $ICSig(D, INT, \mathcal{P}, S)$ and set $\text{Reveal}_\alpha = NULL$.

Claim 7.10 *If D and INT are honest then D will never A-cast S during Ver.*

PROOF: Since D is honest, he will send the verification information (α_i, v_i, r_i) to verifier P_i in $\mathcal{P}$. The honest verifiers (at least $2t+1$) will eventually receive the verification information from D and will inform INT by sending `Received-From-D` signal. Hence, the honest INT will eventually construct $ReceivedSet$ and will correctly A-cast $(d, B(x), ReceivedSet)$ during Ver. So during Ver, D will find $B(\alpha_i) = dv_i + r_i$ for all $P_i \in ReceivedSet$. Thus D will never A-cast S during Ver. □

Lemma 7.11 (AICP-Correctness1) *If D and INT are honest, then $ICSig(D, INT, \mathcal{P}, S)$ produced by INT during Reveal-Public will be accepted by each honest verifier.*

PROOF: For an honest D, $(F(x), R(x))$ held by honest INT and (α_i, v_i, r_i) held by honest verifier P_i in $ReceivedSet$ will satisfy $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$. Moreover by previous claim, D will never A-cast S during Ver. Hence $ICSig(D, INT, \mathcal{P}, S) = F(x)$. Now every honest verifier P_i in $ReceivedSet$ will A-cast `Accept` during Reveal-Public as **C1** i.e $v_i = F(\alpha_i)$ will hold in protocol Reveal-Public. Since there are at least $t+1$ honest verifiers in $ReceivedSet$, $ICSig(D, INT, \mathcal{P}, S)$ will be accepted by every honest verifier. □

Claim 7.12 *If $(F(x), R(x))$ held by an honest INT and (α_i, v_i, r_i) held by an honest verifier $P_i \in ReceivedSet$ satisfy $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$, then except with probability $\frac{\epsilon}{n}$, $B(\alpha_i) \neq dv_i + r_i$.*

PROOF: We first prove that for $(F(x), R(x))$ held by an honest INT and (α_i, v_i, r_i) held by honest verifier $P_i \in ReceivedSet$, there is *only one* non-zero d for which $B(\alpha_i) = dv_i + r_i$, even though $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. For otherwise, assume there exists another non-zero element $e \neq d$, for which $B(\alpha_i) = ev_i + r_i$ is true, even if $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. This implies that $(d-e)F(\alpha_i) = (d-e)v_i$ or $F(\alpha_i) = v_i$, which is a contradiction. Now since d is randomly chosen by honest INT *only after* D handed over $(F(x), R(x))$ to INT and (α_i, v_i, r_i) to every *honest* $P_i \in ReceivedSet$, a corrupted D has to guess d in advance during Gen to make sure that $B(\alpha_i) = dv_i + r_i$ holds. However, D can guess d with probability at most $\frac{1}{|\mathbb{F}|-1} \approx \frac{\epsilon}{n}$. Hence only with probability at most $\frac{\epsilon}{n}$, corrupted D can make $B(\alpha_i) = dv_i + r_i$, even though $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$. □

Lemma 7.13 (AICP-Correctness2) *If an honest INT holds an $ICSig(D, INT, \mathcal{P}, S)$ at the end of* **Verification Phase**, *then $ICSig(D, INT, \mathcal{P}, S)$ will be accepted in Reveal-Public by each honest verifier, except with probability ϵ.*

PROOF: We prove the lemma considering D to be corrupted because when D is honest, the lemma follows from Lemma 7.11. Now the proof can be divided into following two cases:

1. $ICSig(D, INT, \mathcal{P}, S) = S$: In this case, the lemma holds trivially, without any error.

2. $ICSig(D, INT, \mathcal{P}, S) = F(x)$: Here, we show that except with probability ϵ, each honest verifier in $ReceivedSet$ will A-cast `Accept` during Reveal-Public. So let P_i be an honest verifier in $ReceivedSet$. We now have the following cases depending on the relation that holds between the information held by INT (i.e $(F(x), R(x))$) and information held by the honest $P_i \in ReceivedSet$ (i.e (α_i, v_i, r_i)):

(a) $F(\alpha_i) = v_i$: Here P_i will A-cast `Accept` without any error probability as **C1** (i.e $F(\alpha_i) = v_i$) will hold.

(b) $F(\alpha_i) \neq v_i$ *and* $R(\alpha_i) = r_i$: Here P_i will A-cast `Accept` without any error probability, as **C2** (i.e $B(\alpha_i) \neq dv_i + r_i$) will hold.

(c) *If* $F(\alpha_i) \neq v_i$ *and* $R(\alpha_i) \neq r_i$: Here P_i will A-cast `Accept` except with probability $\frac{\epsilon}{n}$, as condition **C2** will hold, except with probability $\frac{\epsilon}{n}$ (see Claim 7.12).

As shown above, there is a negligible error probability of $\frac{\epsilon}{n}$ with which an *honest* $P_i \in ReceivedSet$ may A-cast Reject when $F(\alpha_i) \neq v_i$ and $R(\alpha_i) \neq r_i$ (i.e the third case). This happens if a corrupted D can guess the unique d in Gen, corresponding to P_i and it so happens that INT also selects the same d in Ver and therefore condition **C2** does not hold good for P_i in Reveal-Public. Now D can guess a d_i for each honest verifier P_i in *ReceivedSet* and if it so happens that honest INT chooses d which is same as one of those $t+1$ d_i's guessed by D, then condition **C2** will not be satisfied for the honest verifier P_i for whom $d_i = d$ and therefore P_i will broadcast `Reject`. This may lead to the rejection of $ICSig(D, INT, \mathcal{P}, S)$, as t corrupted verifiers may always broadcast `Reject`. But the above event can happen with error probability $\frac{t+1}{|\mathbb{F}|-1} = (t+1)\frac{\epsilon}{n} \approx \epsilon$. This is because there are $t+1$ d_i's and INT has selected some d randomly from $\mathbb{F} \setminus \{0\}$. This implies that all honest verifiers in *ReceivedSet* will A-cast `Accept` during Reveal, except with error probability ϵ.

This completes the proof of the lemma. □

Lemma 7.14 (AICP-Correctness3) *If D is honest, then during Reveal-Public, with probability at least $(1-\epsilon)$, every $ICSig(D, INT, \mathcal{P}, S')$ with $S' \neq S$ produced by a corrupted INT will not be accepted by an honest verifier.*

Proof: Here again we have two cases. If $ICSig(D, INT, \mathcal{P}, S) = S$, then the lemma holds trivially from the protocol steps. So we now prove the lemma when $ICSig(D, INT, \mathcal{P}, S) = F(x)$. Here a corrupted INT can produce $S' \neq S$ by A-casting $F'(x) \neq F(x)$ during Reveal-Public such that the lower order ℓ coefficients of $F'(x)$ are the elements of S'. We now claim that if INT does so, then except with probability ϵ, every honest verifier P_i in *ReceivedSet* will A-cast `Reject` during Reveal-Public. In the following, we show that the conditions for which an honest verifier P_i in *ReceivedSet* would A-cast `Accept` are either impossible or may happen with probability ϵ:

1. $F'(\alpha_i) = v_i$: Since P_i and D are honest, corrupted INT has no information about α_i, v_i. Hence the probability that INT can ensure $F'(\alpha_i) = v_i = F(\alpha_i)$ is same as the probability with which INT can correctly guess α_i, which is at most $\frac{1}{|\mathbb{F}-1|} \approx \frac{\epsilon}{n}$ (since α_i is randomly chosen by D from $\mathbb{F}$).

2. $B(\alpha_i) \neq dv_i + r_i$: This case is never possible since D is honest. If $B(\alpha_i) \neq dv_i + r_i$ corresponding to $P_i \in ReceivedSet$, then honest D would have A-casted S during Ver and hence $ICSig(D, INT, \mathcal{P}, S)$ would have been equal to S, which is a contradiction to our assumption that $ICSig(D, INT, \mathcal{P}, S) = F(x)$.

As shown above, there is a negligible error probability of $\frac{\epsilon}{n}$ with which an honest P_i may A-cast Accept, even if the corrupted INT produces $F'(x) \neq F(x)$. This happens if the corrupted INT can guess α_i corresponding to honest verifier $P_i \in ReceivedSet$. Now there are $t+1$ honest verifiers in $ReceivedSet$. A corrupted INT can guess α_i for any one of those $t+1$ honest verifiers and thereby can ensure that $F'(\alpha_i) = v_i$ holds for some honest P_i (which in turn implies P_i will A-cast Accept). This will ensure that INT's $ICSig(D, INT, \mathcal{P}, S')$ will be accepted, as t corrupted verifiers may always A-cast Accept. But the above event can happen with probability at most $\frac{t+1}{|\mathbb{F}|-1} = (t+1)\frac{\epsilon}{n} \approx \epsilon$. This asserts that every $ICSig(D, INT, \mathcal{P}, S')$ with $S' \neq S$ revealed by a corrupted INT will be rejected by all honest verifiers with probability at least $(1-\epsilon)$. □

Lemma 7.15 (AICP-Secrecy) *If D and INT are honest and INT has not started Reveal-Public, then $\mathcal{A}_t$ will have no information about S.*

Proof: During Distr, $\mathcal{A}_t$ will know t distinct points on $F(x)$ and $R(x)$. Since both $F(x)$ and $R(x)$ are of degree-$(\ell+t)$, the lower order ℓ coefficients of both $F(x)$ and $R(x)$ are information theoretically secure. During Ver, $\mathcal{A}_t$ will know d and $dF(x) + R(x)$. Since both $F(x)$ and $R(x)$ are random and independent of each other, it still holds that the lower order ℓ coefficients of $F(x)$ is information theoretically secure. Also, if D and INT are honest, then D will never broadcast S during Ver. Hence the lemma. □

Lemma 7.16 (Communication Complexity of MVMS-AICP-II)

- *Protocol Gen privately communicates $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits.*
- *Protocol Ver requires A-cast of $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits and private communication of $\mathcal{O}(n\log n)$ bits.*
- *Protocol Reveal-Private privately communicates $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits.*
- *Protocol Reveal-Public A-casts $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits.*

Proof: In protocol Gen, D privately gives $\ell+t$ field elements to INT and three field elements to each verifier. Since each field element can be represented by $\kappa = \mathcal{O}(\log\frac{1}{\epsilon})$ bits, Gen incurs a private communication of $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits. In protocol Ver, every verifier privately sends `Received-From-D` signal to INT, thus incurring a private communication of $\mathcal{O}(n)$ bits. In addition, INT A-casts $B(x)$ containing $\ell+t$ field elements, thus incurring A-cast of $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits. In protocol Reveal-Public, INT A-casts $F(x)$, consisting of $\ell+t$ field elements, while each verifier A-casts `Accept/Reject` signal. So Reveal-Public involves A-cast of $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits.

In protocol Reveal-Private, INT sends $ICSig(D, INT, \mathcal{P}, S)$ to P_α and individual verifier sends their values to P_α privately. So Reveal-Private requires a private communication of $\mathcal{O}((\ell+n)\log\frac{1}{\epsilon})$ bits. □

Theorem 7.17 *Protocol MVMS-AICP-II is an efficient AICP.*

Proof: The theorem follows from Lemma 7.11, Lemma 7.13, Lemma 7.14, Lemma 7.15 and Lemma 7.16. □

7.5 Discussion About MVMS-AICP-I and MVMS-AICP-II

Now if we consider MVMS-AICP-I with only P_α-private-revelation, then the communication done in the protocol is only private communication (and no A-cast). For our statistical AMPC (asynchronous MPC) with optimal resilience (presented in Chapter 10), we require to design an optimally resilient statistical AVSS with private reconstruction where a specific party (instead of all the parties) is allowed to reconstruct the secret. Moreover, to bound the communication complexity by certain bound, we require the A-cast communication complexity of the AVSS protocol to be independent of ℓ which is the number of secrets shared by the AVSS protocol. Protocol MVMS-AICP-I serves our purpose in this case.

On the other hand, if we consider protocol MVMS-AICP-II with only public revelation, then we see that the protocol has same A-cast and private communication. For our ABA (asynchronous BA) with optimal resilience (presented in Chapter 9), we require an AVSS with public reconstruction that requires an AICP with public revelation. We could use MVMS-AICP-I with public revelation for this purpose. But this requires a communication complexity $\mathcal{O}((\ell + t\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits of A-cast (and private communication) which is more that the A-cast communication of MVMS-AICP-II. Hence for ABA, we will use MVMS-AICP-II. Another purpose of presenting both the AICPs is to display two different techniques to achieve the same task.

7.6 Comparison of MVMS-AICP-I and MVMS-AICP-II with Existing AICP of [39, 35]

As mentioned earlier, there is only one AICP so far [39, 35]. The AICP [39, 35] is nothing but the ICP of [138] adopted in asynchronous network. The protocol is also designed with single verifier and single secret. We may extend it to the case of n verifiers and ℓ secrets easily. In Table 7.1 we list the communication complexity of MVMS-AICP-I, MVMS-AICP-II and the protocol of [39, 35] extended for n verifiers and ℓ secrets.

Table 7.1: Communication Complexity of protocol MVMS-AICP-I, MVMS-AICP-II and Existing AICP of [39, 35] with $n = 3t+1$ verifiers and ℓ secrets.

Ref.	Gen	Ver	Reveal-Public	Reveal-Private
[39, 35]	Private– $\mathcal{O}(\ell n(\log\frac{1}{\epsilon})^2)$	Private– $\mathcal{O}(\ell n(\log\frac{1}{\epsilon})^2)$	—	Private– $\mathcal{O}(\ell n(\log\frac{1}{\epsilon})^2)$
MVMS-AICP-I	Private– $\mathcal{O}((\ell + n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$	Private– $\mathcal{O}((\ell + n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$	A-cast $\mathcal{O}((\ell + n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$	Private– $\mathcal{O}((\ell + n\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$
MVMS-AICP-II	Private– $\mathcal{O}((\ell + n)\log\frac{1}{\epsilon})$	A-cast– $\mathcal{O}((\ell + n)\log\frac{1}{\epsilon})$	A-cast– $\mathcal{O}((\ell + n)\log\frac{1}{\epsilon})$	Private $\mathcal{O}((\ell + n)\log\frac{1}{\epsilon})$

7.7 Definition and Notations for Using MVMS-AICP-I and MVMS-AICP-II as Black Box

We now present the following definition:

Definition 7.18 (IC Signature with ϵ Error) *An IC signature $ICSig(D, INT, \mathcal{P}, S)$ for some secret S, is said to have ϵ error, if it satisfies the following:*

1. **AICP-Correctness1** *without any error;*
2. **AICP-Correctness2** *with error probability of at most ϵ;*
3. **AICP-Correctness3** *with error probability of at most ϵ;*
4. **AICP-Secrecy** *without any error.*

Notice that if an IC signature is generated in MVMS-AICP-I or MVMS-AICP-II (which is executed with error probability ϵ), then the IC signature will have ϵ error. This follows from the proofs of Lemma 7.4, 7.5, 7.6, 7.7 and Lemma 7.11, 7.13, 7.14, 7.15.

We use the following notation for using protocols MVMS-AICP-I and MVMS-AICP-II as black boxes.

Notation 7.19 (Notation for Using MVMS-AICP-I/MVMS-AICP-II) *Recall that D and INT can be any party from $\mathcal{P}$. In the subsequent chapters, we use the following conventions. We say that:*

1. **Gen:** *"P_i sends $ICSig(P_i, P_j, \mathcal{P}, S)$ having ϵ error to P_j" to mean that P_i acting as dealer D and considering P_j as INT, executes $\mathsf{Gen}(P_i, P_j, \mathcal{P}, S, \epsilon)$;*
2. **Ver:** *"P_i receives $ICSig(P_j, P_i, \mathcal{P}, S)$ having ϵ error from P_j" to mean that P_i as INT has received $ICSig(P_j, P_i, \mathcal{P}, S)$ after executing $\mathsf{Ver}(P_j, P_i, \mathcal{P}, S, \epsilon)$;*
3. **Reveal-Private:**
   (a) *"P_i reveals $ICSig(P_j, P_i, \mathcal{P}, S)$ having ϵ error to P_α" to mean P_i as INT executes $\mathsf{Reveal\text{-}Private}(P_j, P_i, \mathcal{P}, S, P_\alpha, \epsilon)$ along with the participation of the verifiers in $\mathcal{P}$;*
   (b) *"P_α completes revelation of $ICSig(P_j, P_i, \mathcal{P}, S)$ with $\mathsf{Reveal}_\alpha = S$" to mean that P_α has successfully completed $\mathsf{Reveal\text{-}Private}(P_j, P_i, \mathcal{P}, S, P_\alpha, \epsilon)$ with $\mathsf{Reveal}_\alpha = S$.*
4. **Reveal-Public:**
   (a) *"P_i reveals $ICSig(P_j, P_i, \mathcal{P}, S)$ having ϵ" to means P_i as INT executes $\mathsf{Reveal\text{-}Public}(P_j, P_i, \mathcal{P}, S, \epsilon)$ along with the participation of the verifiers in $\mathcal{P}$.*
   (b) *"P_k completes revelation $ICSig(P_j, P_i, \mathcal{P}, S)$ with $\mathsf{Reveal}_k = \overline{S}$ " to mean that P_k as a verifier has successfully completed $\mathsf{Reveal\text{-}Public}(P_j, P_i, \mathcal{P}, S, \epsilon)$ with $\mathsf{Reveal}_k = \overline{S}$.*

7.8 Conclusion and Open Problems

In this chapter, we have extended the basic bare-bone definition of AICP, used by Canetti et al. [35] to capture multiple verifiers and multiple secrets concurrently. Then we have presented two AICPs that will be used in two different contexts, namely in AMPC and ABA. We have shown that our AICPs are better than the existing protocol in terms of communication complexity.

We conclude this chapter with an interesting open question:

Open Problem 11 *Can we improve the communication complexity of MVMS-AICP-I and MVMS-AICP-II when there are $n = 3t + 1$ verifiers?*

Probably, if we can get a ICP (in synchronous network) with better complexity than the one presented in Chapter 2 of this thesis, then we can adopt that protocol to obtain a AICP with better complexity. More generally we may try to answer the following question:

Open Problem 12 *What is the communication complexity lower bound for AICP with $n = 3t + 1$ verifiers?*

Chapter 8

Efficient Statistical AVSS Protocols With Optimal Resilience

An AVSS is a two phase (Sharing, Reconstruction) protocol carried out among n parties in the presence of a *computationally unbounded active* adversary, who can corrupt up to t parties. We assume that every two parties in the network are directly connected by a pairwise secure channel.

In this chapter, we present two novel statistical AVSS protocols with *optimal resilience*; i.e. with $n = 3t + 1$. The protocols are designed to attain different properties and are used in different contexts. While one of the protocols is used in our ABA protocol with optimal resilience (presented in Chapter 9), the other one is used in our AMPC with optimal resilience (presented in Chapter 10). Also, it is important to note that the protocols are based on completely disjoint techniques. There is only one statistical AVSS protocol with $n = 3t + 1$ in the literature reported in [39]. Both our AVSS protocols show significant improvement over the AVSS of [39] in terms of the communication complexity.

As a key tool for our statistical AVSS protocols, we construct protocols for weaker notion of statistical AVSS called statistical asynchronous Weak Secret Sharing or statistical AWSS in short. Our statistical AWSS protocols use the AICPs presented in Chapter 7 as building blocks.

8.1 Introduction

Over the last three decades, active research has been carried out on VSS by several researchers, and many interesting and significant results have been obtained dealing with high efficiency, security against general adversaries, security against mixed types of corruptions, long-term security, provable security, etc (see [43, 55, 108, 9, 95, 20, 41, 62, 63, 137, 48, 21, 39, 138, 73, 91, 93, 109, 125, 12, 14, 98, 126, 50, 47, 35, 96, 28, 133, 66, 64, 8, 37, 22, 53, 92, 123, 145, 34, 97] and their references). However, almost all of these solutions are for the synchronous model, where it is assumed that every message in the network is delayed at most by a given constant. This assumption is very strong because a single delayed message would completely break down the overall security of the protocol. Therefore, VSS protocols for the synchronous model are not suited for real world networks like the Internet.

Hence a new line of research on VSS over asynchronous network has been initiated. It is well known that asynchronous network models Internet more appropriately than synchronous network. VSS protocols that are designed to work over asynchronous network is called Asynchronous VSS or AVSS.

8.1.1 The Network and Adversary Model

In this chapter, we follow the network model of [35]. Specifically, we assume that an AVSS protocol is carried out among a set of n parties, say $\mathcal{P} = \{P_1, \ldots, P_n\}$, where every two parties are directly connected by a secure channel and t out of the n parties can be under the influence of a *computationally unbounded Byzantine (active) adversary*, denoted as $\mathcal{A}_t$. We assume $\mathcal{A}_t$ to be *rushing* [125, 91, 48], who may choose to first listen all the messages sent to the corrupted parties by the honest parties, before allowing the corrupted parties to send their messages. The parties not under the influence of $\mathcal{A}_t$ are called *honest or uncorrupted*. We assume that there is a specific party in $\mathcal{P}$, called the *dealer* D, who wants to share the secret in AVSS protocol. Lastly in this chapter, we assume $n = 3t + 1$.

The underlying network is asynchronous, where the communication channels between the parties have arbitrary, yet finite delay (i.e the messages are guaranteed to reach eventually). To model this, $\mathcal{A}_t$ is given the power to schedule the delivery of *all* messages in the network. However, $\mathcal{A}_t$ can only schedule the messages communicated between honest parties, without having any access to the contents of the message. In asynchronous network, the inherent difficulty in designing a protocol comes from the fact that when a party does not receive an expected message then he cannot decide whether the sender is corrupted (and did not send the message at all) or the message is just delayed in the network. So a party can not wait to consider the values sent by all parties, as waiting for all of them could turn out to be endless. Hence the values of up to t (potentially honest) parties may have to be ignored. Due to this the protocols in asynchronous network are generally involved in nature and require new set of primitives. For an comprehensive introduction to asynchronous protocols, see [35].

8.1.2 Definitions

Informally, any AVSS scheme consists of a pair of protocols (Sh, Rec). Protocol Sh [1] allows a special party called *dealer* (denoted as D), to share a secret $s \in \mathbb{F}$ (an element from a finite field $\mathbb{F}$) among a set of n parties in a way that allows for a unique reconstruction of s by every party using protocol Rec [2]. Moreover, if D is *honest*, then the secrecy of s is preserved till the end of Sh. Any AWSS protocol relaxes the strict condition of reconstructing the secret s to reconstructing either the secret or NULL. Now we start with the formal definition of statistical AWSS.

Definition 8.1 (Statistical AWSS) *Let (Sh, Rec) be a pair of protocols in which a dealer $D \in \mathcal{P}$ shares a secret s using Sh among the n parties in $\mathcal{P}$. We say that (Sh, Rec) is an (n, t) statistical AWSS scheme if all the following hold:*

- **Termination**: *With probability at least $1 - \epsilon$, the following requirements hold:*

[1] Sh is the protocol for sharing phase of AVSS scheme

[2] Rec is the protocol for reconstruction phase of AVSS scheme

1. *If D is honest then each honest party will eventually terminate protocol* Sh.
2. *If some honest party has terminated protocol* Sh*, then irrespective of the behavior of D, each honest party will eventually terminate* Sh.
3. *If all honest parties have terminated* Sh *and invoked* Rec*, then each honest party will eventually terminate* Rec.

- **Correctness**: *With probability at least $1 - \epsilon$, the following requirements hold:*

  1. **Correctness 1 (AWSS)**: *If D is honest then each honest party upon terminating* Rec*, outputs the shared secret s.*
  2. **Correctness 2 (AWSS)**: *If D is faulty and some honest party has terminated* Sh*, then there exists a unique $s' \in \mathbb{F} \cup \{NULL\}$, such that each honest party upon terminating* Rec *will output either s' or $NULL$. This property is also called as* weak-commitment.

- **Secrecy**: *If D is honest and no honest party has begun executing protocol* Rec*, then $\mathcal{A}_t$ has no information about s.*

The definition of statistical AWSS does not stop some honest party to reconstruct the committed secret s' and some other honest party to reconstruct $NULL$, when D is corrupted.

We now present two different definitions of statistical AVSS: strong definition of statistical AVSS (parallel to strong definition of statistical VSS in synchronous network (see Definition 3.4 in Chapter 3)) and weak definition of AVSS (parallel to the weak definition of statistical VSS in synchronous network (see Definition 3.3 in Chapter 3)).

Definition 8.2 (Strong definition of Statistical AVSS [19, 35]) *It is same as statistical AWSS except that* ***Correctness 2 (AWSS)*** *property is strengthened as follows:*

- **Correctness 2 (AVSS)**: *If D is corrupted and some honest party has terminated* Sh*, then there exists a fixed $s' \in \mathbb{F}$, such that each honest party upon completing* Rec*, will output only s'.*

Definition 8.3 (Weak definition of Statistical AVSS) *It is same as statistical AWSS except that* ***Correctness 2 (AWSS)*** *property is strengthened as follows:*

- **Correctness 2 (AVSS)**: *If D is corrupted and some honest party has terminated* Sh*, then there exists a fixed $s' \in \mathbb{F} \cup NULL$, such that each honest party upon completing* Rec*, will output only s'.*

Remark 8.4 *So far in the literature of AVSS, weak definition of AVSS was never introduced and used. It is the strong definition of AVSS which was prevalent. But in this thesis, since we design protocol for both types, we felt that it is important to distinguish between these two notions. In the sequel, we call an AVSS as strong AVSS when it satisfies strong definition of AVSS and likewise we call an AVSS as weak AVSS when it satisfies the weak definition of AVSS.*

The difference between Strong and Weak Statistical AVSS: The difference between strong and weak statistical AVSS is as follows: In weak statistical AVSS, a corrupted dealer D may get away with not committing a value/ secret from field $\mathbb{F}$; but in a strong statistical AVSS D is forced to commit a secret from $\mathbb{F}$. In this thesis, we will show that weak statistical AVSS is enough for constructing ABA, whereas we require strong statistical AVSS for designing AMPC protocol. In case of weak statistical AVSS, we fix a predefined default value $s^\star \in \mathbb{F}$ and when D commits $NULL$, then in reconstruction phase every party assumes $s^\star$ as the D's committed secret. That is how we may interpret that D has committed some secret from $\mathbb{F}$. But as mentioned earlier, weak definition of AVSS is not sufficient for AMPC (more discussion follows in subsequent chapters). $\diamondsuit$

The above definitions of AWSS and AVSS can be extended for secret S containing multiple elements (say ℓ with $\ell > 1$) from $\mathbb{F}$.

Remark 8.5 (AWSS and AVSS with Private Reconstruction) *The definitions of AWSS and AVSS as given above consider "public reconstruction", where all parties publicly reconstruct the secret in* Rec*. A common variant of these definitions consider "private reconstruction", where only some specific party, say $P_\alpha \in \mathcal{P}$, is allowed to reconstruct the secret in* Rec*.*

In this chapter, we present our protocols with both public and private reconstruction.

8.1.3 Contribution of This Chapter

From [39], statistical AVSS tolerating $\mathcal{A}_t$ is possible iff $n \geq 3t + 1$. Therefore, any statistical AVSS with $n = 3t + 1$ parties is said to have *optimal resilience.* The only known statistical AVSS protocol with optimal resilience is due to [39]. The AVSS scheme was designed to be used for constructing ABA protocol.

In this chapter, we present two new statistical AVSS schemes with optimal resilience. Our protocols are designed with both public as well as private reconstruction. One protocol satisfies weak definition of AVSS and the other one satisfies strong definition of AVSS. In Table 8.1, we compare the communication complexity of our AVSS protocols with the AVSS of [39, 35]. The AVSS corresponding to the middle row refers to the weak statistical AVSS and the AVSS corresponding to the last row refers to the strong statistical AVSS.

As shown in Table 8.1, our AVSS protocols attain significantly better communication complexity than the AVSS of [39] for any value of ℓ.

Later we will show that the AVSS satisfying the definition of weak statistical AVSS is sufficient for designing ABA and thus we will use our weak statistical AVSS for constructing our ABA presented in Chapter 9. However, to be applicable for AMPC, we require that AVSS should be strong statistical AVSS. Hence we use our strong statistical AVSS for constructing statistical AMPC protocol presented in Chapter 10.

In order to design our AVSS protocols, we first propose a new AWSS protocol that uses AICP as black box. By using MVMS-AICP-I and MVMS-AICP-II (presented in Chapter 7) separately in the AWSS protocol, we obtain two different AWSS protocols which are further used to design two different AVSS schemes.

Table 8.1: Comparison of our AVSS protocols with the exiting AVSS Protocol of [39, 35] in terms of Communication Complexity.

Ref.	Sharing Phase	Private Reconstruction	Public Reconstruction	# Secrets
[39] ⋆	Private† $\mathcal{O}(n^9(\log \frac{1}{\epsilon})^4)$ A-cast– $\mathcal{O}(n^9(\log \frac{1}{\epsilon})^2 \log n)$	– – – –	Private– $\mathcal{O}(n^6(\log \frac{1}{\epsilon})^3)$ A-cast– $\mathcal{O}(n^6 \log \frac{1}{\epsilon} \log n)$	1
This chapter	Private– $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ A-cast– $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$	Private– $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ A-cast– NIL	Private– NIL A-cast– $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$	ℓ
This chapter	Private– $\mathcal{O}((\ell n^3 + n^4 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ A-cast– $\mathcal{O}(n^3 \log(n))$	Private– $\mathcal{O}((\ell n^3 + n^4 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ A-cast– NIL	Private– NIL A-cast– $\mathcal{O}((\ell n^3 + n^4 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$	ℓ

⋆ Since the communication complexity analysis of the AVSS of [39] was never done before, we do the same in section 8.7 of this chapter for the sake of completeness.

† Communication over private channels between pair of parties in $\mathcal{P}$.

To bound the error probability by ϵ, our AVSS protocols work over a finite Galois field $\mathbb{F}$ with $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the value of ϵ and the relation between ϵ and κ. The exact relationship between κ and ϵ will be different for two AVSS protocols and hence they are provided in respective sections. We assume that $\ell = poly(\kappa, n)$. For both the protocols, each field element from field $\mathbb{F}$ can be represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

In order to bound the error probability of any of our AVSS protocol by some specific value of ϵ, we find out the *minimum* value of κ that satisfies the relation between κ and ϵ for that protocol. The value for κ will consequently determine the field $\mathbb{F}$ over which the protocol should work.

8.1.4 The Road-map

In section 8.2, we briefly present the approaches used by the only known statistical AVSS protocol of [39] and the approaches used by our protocols. In section 8.3, we present our AWSS protocol. In section 8.4 and 8.5, we present our AVSS protocols. We conclude this chapter with concluding remarks and open problems in section 8.6. Since the communication complexity analysis of the AVSS of [39] was never done before, we do the same in section 8.7.

8.2 Discussion on the Approaches used in the AVSS of [39] and the Approaches used by our AVSS Protocols

In the following, we summarize the approaches used by the AVSS of [39] and the approaches used by our protocols.

1. **Approach of [39]:** The authors of [39] have presented a series of protocols for designing their AVSS scheme. They first designed AICP which is used as a black box for another primitive *Asynchronous Recoverable Sharing* (A-RS). Subsequently, using A-RS, the authors have designed an AWSS

scheme, which is further used to design a variation of AWSS called *Two & Sum AWSS*. Finally using their *Two & Sum AWSS*, an AVSS scheme was presented. Pictorially, the route taken by AVSS scheme of [39] is as follows: *AICP* $\rightarrow$ *A-RS* $\rightarrow$ *AWSS* $\rightarrow$ *Two & Sum AWSS* $\rightarrow$ *AVSS*. Since the AVSS scheme is designed on top of so many sub-protocols, it becomes highly communication intensive as well as very much involved. The scheme requires a private communication of $\mathcal{O}(n^9(\log \frac{1}{\epsilon})^4)$ bits and A-cast of $\mathcal{O}(n^9(\log \frac{1}{\epsilon})^2 \log(n))$ bits to share a *single* element from $\mathbb{F}$ (see the first row of Table 8.1).

2. **Approach of This Thesis:** We used the following simpler route to design our AVSS schemes: *AICP* $\rightarrow$ *AWSS* $\rightarrow$ *AVSS*. Moreover, due to the new design approach used in our AICP, AWSS and AVSS protocols, our AVSS protocols provide much better communication complexity than the AVSS of [39] (as shown in last two rows of Table 8.1).

8.3 Statistical AWSS Protocol

For the sake of simplicity, we first present our AWSS protocol sharing a single secret and then extend the protocol for multiple (i.e ℓ) secrets. We will later show that dealing with multiple secrets concurrently in a protocol provides with better communication complexity than multiple executions of protocol dealing with single secret. In our protocol, we use IC signatures in such a way that the AICP can be replaced by either MVMS-AICP-I or MVMS-AICP-II (presented in Chapter 7). Depending on which AICP is used will finally decide on the field $\mathbb{F}$ over which all the computation of our AWSS should be carried out. Hence for the time being, let us concentrate on the bare-bone structure of our AWSS and later we will derive two protocols out of it by replacing the underlying AICP by MVMS-AICP-I and MVMS-AICP-II.

8.3.1 AWSS Scheme for Sharing a Single Secret

We now present a novel AWSS scheme with $n = 3t+1$ called AWSS, consisting of sub-protocols (AWSS-Share, AWSS-Rec-Private, AWSS-Rec-Public). While AWSS-Share allows D to share a secret s, AWSS-Rec-Private enables private reconstruction of s or $NULL$ by a specific party, say $P_\alpha \in \mathcal{P}$ and likewise AWSS-Rec-Public enables public reconstruction of either D's shared secret or $NULL$. We call the private reconstruction as *P_α-weak-private-reconstruction*. Moreover, if D is *corrupted*, then s can be either from $\mathbb{F}$ or it can be $NULL$ (in a sense explained in the sequel). Our AWSS scheme is somewhat inspired by the WSS scheme of [48] in synchronous settings, with several new ideas added to it, to deal with the asynchrony of the network.

High Level Description of AWSS-Share: We follow the general strategy used in [20, 48, 91, 73, 109] for synchronous settings for sharing the secret s with a symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y, where each party P_i gets the univariate polynomial $f_i(x) = F(x, i)$. In particular, in AWSS-Share, D chooses a symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y such that $F(0, 0) = s$. D then hands over $ICSig(D, INT, \mathcal{P}, f_i(j))$ for every $j = 1, \ldots, n$ to P_i. This step implicitly implies that P_i will receive $f_i(x)$ from D. After receiving these IC signatures from D, the parties then exchange IC signature on their common values (a pair (P_i, P_j) has one common value, namely $F(i,j)$; P_i has $f_i(j)$ and P_j has $f_j(i)$ where $F(i,j) = f_i(j) = f_j(i)$). Then D, in conjunction with all other parties, perform a sequence of communication and computation. As a result of this, at the end of AWSS-Share, every party agrees on a set of $2t+1$ parties, called $WCORE$, such that every party $P_j \in WCORE$ is *IC-committed* to $f_j(0)$ using $f_j(x)$ to a set of $2t+1$ parties, called as OKP_j. P_j is *IC-committed* to $f_j(0)$ using $f_j(x)$ among the parties in OKP_j only when every $P_k \in OKP_j$ received (a) $ICSig(D, P_k, \mathcal{P}, f_k(j))$ and (b) $ICSig(P_j, P_k, \mathcal{P}, f_j(k))$ and ensures $f_k(j) = f_j(k)$ (this should ideally hold due to the selection and distribution of symmetric bivariate polynomial). In some sense, we may view this as every $P_j \in WCORE$ is attempting to commit his received (from D) polynomial $f_j(x)$ among the parties in OKP_j (by giving his *IC Signature* on one point of $f_j(x)$ to each party) and the parties in OKP_j allowing him to do so after verifying that they have got D's IC signature on the same value of $f_j(x)$. We will show that later in the reconstruction phase, every honest P_j's (in $WCORE$) *IC-commitment* will be reconstructed correctly irrespective of whether D is honest or corrupted. Moreover, a corrupted P_j's *IC-commitment* will be reconstructed correctly when D is honest. But on the other hand, a corrupted P_j's *IC-commitment* can be reconstructed to any value when D is corrupted. These properties are at the heart of our AWSS protocol.

Achieving the agreement (among the parties) on $WCORE$ and corresponding OKP_js is a bit tricky in asynchronous network. Even though these sets are constructed on the basis of information that are A-casted by parties, parties may end up with different versions of $WCORE$ and OKP_j's while attempting to generate them locally, due to the asynchronous nature of the network. We solve this problem by asking D to construct $WCORE$ and OKP_js based on A-casted information and then ask D to A-cast the same. After receiving $WCORE$ and OKP_js from the A-cast of D, individual parties ensure the validity of these sets by receiving the same A-cast using which D would have formed these sets. A similar approach was used in the protocols of [1]. Protocol AWSS-Share is formally presented in Fig. 8.1.

Before moving into the discussion and description of AWSS-Rec-Private and AWSS-Rec-Public, we now define what we call as D's AWSS-commitment.

Remark 8.6 (D's AWSS-commitment) *We say that D is AWSS-committed to a secret $s \in \mathbb{F}$ in AWSS-Share if there is a unique degree-t univariate polynomial $f(x)$ such that $f(0) = s$ and every honest P_i in $WCORE$ receives $f(i)$ from D and IC-commits to $f(i)$ among the parties in OKP_i. Otherwise, we say that D has committed $NULL$. An honest D always commits s from $\mathbb{F}$ as in this case $f(x)$ is $f_0(x)(= F(x,0))$, where $F(x,y)$ is the symmetric bivariate polynomial of degree-t in x and y chosen by honest D. Moreover, every honest party P_i in $WCORE$ receives $f(i) = f_0(i)$ which is same as $f_i(0)$ (this can be obtained from $f_i(x)$). But AWSS-Share can not ensure that corrupted D also commits $s \in \mathbb{F}$. This means that a corrupted D may distribute information to the parties such that, polynomial $f_0(x)$ defined by the $f_0(i)(= f_i(0))$ values possessed by honest P_i's in $WCORE$ may not be a degree-t polynomial. In this case we say D is AWSS-committed to $NULL$.*

Our discussion in the sequel will show that for a corrupted D, irrespective of the behavior of the corrupted parties, either D's *AWSS-committed* secret s

Figure 8.1: Sharing Phase of Protocol AWSS for single secret s with $n = 3t + 1$

Protocol AWSS-Share($D, \mathcal{P}, s, \epsilon$)

Distribution: Code for D – Only D executes this code.

1. Select a random, symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y, such that $F(0, 0) = s$. For $i = 1, \ldots, n$, let $f_i(x) = F(x, i)$.
2. For $i = 1, \ldots, n$, send $ICSig(D, P_i, \mathcal{P}, f_i(j))$ to P_i having $\epsilon' = \frac{\epsilon}{n^2}$ error for each $j = 1, \ldots, n$ (Recall the notations for using our AICPs i.e Notation 7.19 in Chapter 7).

Verification: Code for P_i – Every party including D executes this code.

1. Wait to receive $ICSig(D, P_i, \mathcal{P}, f_i(j))$ having ϵ' error for each $j = 1, \ldots, n$ from D.
2. Check if $(f_i(1), \ldots, f_i(n))$ defines degree-t polynomial. If yes then send $ICSig(P_i, P_j, \mathcal{P}, f_i(j))$ to P_j having ϵ' error for all $j = 1, \ldots, n$.
3. If $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$ having ϵ' error, is received from P_j and if $f_i(j) = f_j(i)$, then A-cast $\texttt{OK}(P_i, P_j)$.

WCORE Construction : Code for D – Only D executes this code.

1. For each P_j, build a set $OKP_j = \{P_k | D$ receives $\texttt{OK}(P_k, P_j)$ from the A-cast of $P_k\}$. When $|OKP_j| = 2t + 1$, then P_j's *IC-commitment* on $f_j(0)$ is over (or we may say that P_j is *IC-committed* to $f_j(0)$) and add P_j in $WCORE$ (which is initially empty).
2. Wait until $|WCORE| = 2t + 1$. Then A-cast $WCORE$ and OKP_j for all $P_j \in WCORE$.

WCORE Verification & Agreement on WCORE : Code for P_i

1. Wait to obtain $WCORE$ and OKP_j for all $P_j \in WCORE$ from D's A-cast, such that $|WCORE| = 2t + 1$ and $|OKP_j| = 2t + 1$ for each $P_j \in WCORE$.
2. Wait to receive $\texttt{OK}(P_k, P_j)$ for all $P_k \in OKP_j$ and $P_j \in WCORE$. After receiving all these $\texttt{OK}$s, accept the $WCORE$ and OKP_j's received from D and terminate AWSS-Share.

(which belongs to $\mathbb{F} \cup \{NULL\}$) or $NULL$ will be reconstructed by each honest party in protocol AWSS-Rec-Private and AWSS-Rec-Public.

High Level Idea of AWSS-Rec-Private & AWSS-Rec-Public: In the reconstruction phase, the parties in $WCORE$ and corresponding OKP_j's are used in order to reconstruct D's AWSS-committed secret. Precisely, for every $P_j \in WCORE$, P_j's *IC-commitment* ($f_j(0)$) is reconstructed by asking every party $P_k \in OKP_j$ to reveal $ICSig(D, P_k, \mathcal{P}, f_k(j))$ and $ICSig(P_j, P_k, \mathcal{P}, f_j(k))$ such that $f_k(j) = f_j(k)$ holds. Since there are at least $t + 1$ honest parties in OKP_j, eventually at least $t+1$ $f_j(k)$'s and $f_k(j)$'s will be revealed with which $f_j(x)$ and thus $f_j(0)$ will be reconstructed. Then $f_j(0)$'s are used to construct the univariate polynomial $f_0(x)$ that is committed by D during AWSS-Share.

Asking $P_k \in OKP_j$ to reveal D's IC signature ensures that when D is *honest*, then even for a *corrupted* $P_j \in WCORE$, the reconstructed polynomial $f_j(x)$ will be same as the one handed over by D to P_j in sharing phase (that is a corrupted P_j's IC-commitment $f_j(0)$ will be reconstructed correctly). This helps our AWSS protocol to satisfy **Correctness 1** property of AWSS. Now asking P_k in OKP_j to reveal P_j's signature ensures that even if D is *corrupted*, for an *honest* $P_j \in WCORE$, the reconstructed polynomial $f_j(x)$ will be same as the one received by P_j from D in AWSS-Share (that is an honest P_j's IC-commitment $f_j(0)$ will be reconstructed correctly even though D is corrupted). This helps to ensure **Correctness 2** property. Summing up, when at least one of D and P_j is honest, P_j's *IC-commitment* (i.e $f_j(0)$) will be revealed properly. But when both D and P_j are corrupted, P_j's *IC-Commitment* can be revealed as any $\overline{f_j}(0)$ which may or may not be equal to $f_j(0)$. It is the later property that makes our protocol to qualify as a AWSS protocol rather than a AVSS protocol. Protocol AWSS-Rec-Private and AWSS-Rec-Public is formally given in Fig. 8.2.

We now prove the properties of our AWSS scheme, considering AWSS-Rec-Public as the reconstruction phase protocol. The proofs can be twisted in a straight forward manner for the case when AWSS-Rec-Private is considered as the reconstruction phase protocol.

Lemma 8.7 (AWSS-Termination) *Protocol AWSS satisfies* **termination** *property.*

PROOF:

- **Termination 1:** When D is honest then eventually all honest parties will receive desired IC signatures from D and will also eventually exchange IC signatures on their common values and will A-cast OK for each other. Hence every honest P_j will eventually complete his *IC-commitment* on $f_j(0)$ with at least $2t+1$ honest parties in OKP_j. So D will eventually include $2t+1$ parties in $WCORE$ (of which at least $t+1$ are honest) and A-cast the same. Now by the property of A-cast, each honest party will eventually receive $WCORE$ from the A-cast of D. Finally, since honest D had included P_j in $WCORE$ after receiving the OK signals from the parties in OKP_j's, each honest party will also receive the same and will eventually terminate AWSS-Share.

- **Termination 2:** If an honest P_i has terminated AWSS-Share, then he must have received $WCORE$ and OKP_j's from the A-cast of D and verified their validity. By properties of A-cast, each honest party will also receive the same and will eventually terminate AWSS-Share.

- **Termination 3:** Since each instance of AICP is executed with an error probability $\epsilon' = \frac{\epsilon}{n^2}$, if P_i (acting as INT) is honest and has received an IC signature, then IC signature produced by P_i during Reveal-Public will be accepted by every honest party without any error probability when D is honest (by **AICP-Correctness1** and except with probability ϵ' when D

Figure 8.2: Reconstruction Phase of AWSS Scheme for single secret s with $n = 3t + 1$

Protocol AWSS-Rec-Private($D, \mathcal{P}, s, P_\alpha, \epsilon$):
P_α-weak-private-reconstruction of s

Signature Revelation: Code for P_i — Every party executes this code

1. If P_i belongs to OKP_j for some $P_j \in WCORE$, then privately reveal $ICSig(D, P_i, \mathcal{P}, f_i(j))$ and $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$ to P_α, each having ϵ' error.

Local Computation: Code for P_α — Only P_α executes this code

1. For every $P_j \in WCORE$, reconstruct P_j's *IC-commitment*, say $\overline{f_j}(0)$ as follows:
   (a) Construct a set $ValidP_j = \emptyset$.
   (b) Add $P_k \in OKP_j$ to $ValidP_j$ if the following conditions hold:
      i. Revelation of $ICSig(D, P_k, \mathcal{P}, f_k(j))$ and $ICSig(P_j, P_k, \mathcal{P}, f_j(k))$ are completed with $\mathsf{Reveal}_\alpha = \overline{f_k}(j)$ and $\mathsf{Reveal}_\alpha = \overline{f_j}(k)$; and
      ii. $\overline{f_k}(j) = \overline{f_j}(k)$.
   (c) Wait until $|ValidP_j| = t+1$. Construct a polynomial $\overline{f_j}(x)$ passing through the points $(k, \overline{f_j}(k))$ where $P_k \in ValidP_j$. Associate $\overline{f_j}(0)$ with $P_j \in WCORE$.
2. Wait for $\overline{f_j}(0)$ to be reconstructed for every P_j in $WCORE$.
3. Check whether the points $(j, \overline{f_j}(0))$ for $P_j \in WCORE$ lie on a unique degree-t polynomial $\overline{f_0}(x)$. If yes, then set $\overline{s} = \overline{f_0}(0)$ and terminate AWSS-Rec-Private. Else set $\overline{s} = NULL$ and terminate AWSS-Rec-Private.

AWSS-Rec-Public($D, \mathcal{P}, s, \epsilon$)

Signature Revelation: Code for P_i — Every party executes this code

1. If P_i belongs to OKP_j for some $P_j \in WCORE$, then publicly reveal $ICSig(D, P_i, \mathcal{P}, f_i(j))$ and $ICSig(P_j, P_i, \mathcal{P}, f_j(i))$, each having ϵ' error.

Local Computation: Code for P_i — Every party executes this code
Same as the Local Computation of P_α in AWSS-Rec-Private.

is corrupted (by **AICP-Correctness2**). Since for every $P_j \in WCORE$, $|OKP_j| = 2t + 1$, there are at least $t + 1$ honest parties in OKP_j and each of them may be present in $ValidP_j$ except with probability ϵ'. Thus except with probability $n^2\epsilon' = \epsilon$, P_j's *IC-commitment* will be reconstructed for all $P_j \in WCORE$. Thus except with probability ϵ, each *honest* party will terminate AWSS-Rec-Public after executing remaining steps of **Local Computation**. □

Lemma 8.8 (AWSS-Secrecy) *AWSS satisfies* **secrecy** *property.*

PROOF: We have to consider the case when D is honest. The proof follows from the secrecy of our AICP protocol and properties of symmetric bivariate polynomial of degree-t in x and y [46]. Specifically, without loss of generality, let $P_1, \ldots, P_t$ be under the control of $\mathcal{A}_t$. So during the execution of AWSS-Share, $\mathcal{A}_t$ will know $f_1(x), \ldots, f_t(x)$ and t points on $f_{t+1}(x), \ldots, f_n(x)$. However, $\mathcal{A}_t$ still lacks one more point to uniquely interpolate $F(x, y)$. Hence, $s = F(0, 0)$ will be information theoretically secure. □

Lemma 8.9 (AWSS-Correctness) *Protocol AWSS satisfies* **correctness** *property.*

PROOF:

- **Correctness 1:** Here we have to consider the case when D is *honest*. We show that D's *AWSS-commitment* will be reconstructed correctly except with probability ϵ. We prove the lemma by showing that when D is *honest*, P_j's *IC-commitment* $f_j(0)$ will be correctly reconstructed with probability at least $(1 - \frac{\epsilon}{n})$ for every $P_j \in WCORE$, irrespective of whether P_j is honest or corrupted. Consequently, as $|WCORE| = 2t + 1$, all the honest parties will reconstruct $f_0(x) = F(x, 0)$ and hence the secret $s = f_0(0)$ with probability at least $(1 - (2t + 1)\frac{\epsilon'}{n}) \approx (1 - \epsilon)$. So we consider the following two cases:

  1. Consider an honest P_j in $WCORE$. From **AICP-Correctness3**, a corrupted $P_k \in OKP_j$ can successfully produce $ICSig(P_j, P_k, \mathcal{P}, \overline{f_j}(k))$ such that $\overline{f_j}(k) \neq f_j(k)$, with probability at most ϵ'. As there can be at most t corrupted parties in $ValidP_j$, except with probability $t\epsilon' = \frac{\epsilon}{n}$, the value $\overline{f_j}(k)$ is same as $f_j(k)$ for all $P_k \in ValidP_j$. Hence honest P_j's *IC-commitment* $f_j(0)$ will be correctly reconstructed with probability at least $(1 - \frac{\epsilon}{n})$.
  2. Consider a corrupted P_j in $WCORE$. Now a corrupted $P_k \in OKP_j$ will be able to produce $ICSig(D, P_k, \mathcal{P}, \overline{f_k}(j))$ such that $\overline{f_k}(j) \neq f_k(j)$, with probability ϵ' due to **AICP-Correctness3**. Thus except with probability $t\epsilon' = \frac{\epsilon}{n}$, corresponding to each corrupted $P_j \in WCORE$, the parties in $ValidP_j$ have produced correct points on $f_j(x)$.

- **Correctness 2:** Here we consider the case, when D is *corrupted*. Now there are two cases: (a) D's *AWSS-committed* secret s belongs to $\mathbb{F}$; (b) D's *AWSS-committed* secret s is $NULL$. Whatever may be case, we show that except with probability ϵ, each honest party will either reconstruct s or $NULL$.

  1. We first consider the case when $s \in \mathbb{F}$. This implies that the $f_j(0)$ values received by the honest P_j's in $WCORE$ lies on a degree-t polynomial $f_0(x)$. Moreover every honest P_j in $WCORE$ is *IC-committed* to $f_j(0)$. We now show that in AWSS-Rec-Public, *IC-commitment* of all honest parties in $WCORE$ will be reconstructed correctly with probability at least $(1 - \epsilon)$. So let P_j be an honest party in $WCORE$. Now from **AICP-Correctness3**, a corrupted $P_k \in OKP_j$ can not produce $ICSig(P_j, P_k, \mathcal{P}, \overline{f_j}(k))$ such that $\overline{f_j}(k) \neq f_j(k)$ with probability at

least $(1-\epsilon')$. Hence for honest P_j, $f_j(x)$ and thus $f_j(0)$ will be reconstructed correctly with probability at least $(1-(t+1)\epsilon') \approx (1-\frac{\epsilon}{n})$. As there are at least $t+1$ honest parties in $WCORE$, the probability that the above event happens for all honest parties in $WCORE$ is $(1-\epsilon)$. But for a *corrupted* P_j in $WCORE$, P_j's *IC-commitment* can be revealed to any value $\overline{f_j}(0)$. This is because a corrupted $P_k \in OKP_j$ can produce a valid signature of P_j on any $\overline{f_j}(k)$ as well as a valid signature of D (who is corrupted as well) on $\overline{f_k}(j) = \overline{f_j}(k)$. Also the adversary can delay the messages such that the values of corrupted $P_k \in OKP_j$ are revealed (to parties) before the values of honest parties in OKP_j. Now if reconstructed $\overline{f_j}(0) = f_j(0)$ for all corrupted $P_j \in WCORE$, then s will be reconstructed. Otherwise, $NULL$ will be reconstructed. However, since for all the honest parties of $WCORE$, *IC-commitment* will be reconstructed correctly with probability at least $(1-\epsilon)$ (who in turn define $f_0(x)$), no other secret (other than s) can be reconstructed.

2. We next consider the second case when D's AWSS-committed secret is $NULL$. This implies that the points $(j, f_j(0))$ corresponding to honest P_j's in $WCORE$ do not define a unique degree-t polynomial. It is easy to see that in this case, irrespective of the behavior of the corrupted parties $NULL$ will be reconstructed. This is because the points $f_j(0)$ corresponding to each honest $P_j \in WCORE$ will be reconstructed correctly except with probability ϵ (following the argument given in previous case).

□

Theorem 8.10 *Protocol AWSS is a valid statistical AWSS scheme with $n = 3t+1$ for a single secret.*

PROOF: The proof follows from Lemma 8.7, Lemma 8.8 and Lemma 8.9. □

8.3.1.1 Important Notation

The following notation will be used in our AVSS protocols irrespective of which AICP is used to generate the underlying IC signatures in protocol AWSS.

Notation 8.11 *In our AVSS schemes,*

- *We will invoke AWSS-Share as AWSS-Share($D, \mathcal{P}, f(x), \epsilon$) to mean that D commits to $f(x)$ in AWSS-Share. Essentially here D is asked to choose a symmetric bivariate polynomial $F(x,y)$ of degree-t in x and y, where $F(x,0) = f(x)$ holds. D then tries to give $F(x,i)$ and hence $F(0,i) = f(i)$ to party P_i.*

- *AWSS-Rec-Private will be invoked as AWSS-Rec-Private($D, \mathcal{P}, f(x), P_\alpha, \epsilon$) to enable the P_α-weak-private-reconstruction of $f(x)$.*

- *AWSS-Rec-Public will be invoked as AWSS-Rec-Public($D, \mathcal{P}, f(x), \epsilon$), which allows the parties to reconstruct either $f(x)$ or $NULL$.*

8.3.2 AWSS Scheme for Sharing Multiple Secrets

In this section, we extend protocol AWSS to AWSS-MS consisting of sub-protocols (AWSS-MS-Share, AWSS-MS-Rec-Private, AWSS-MS-Rec-Public) [3]. Protocol AWSS-MS-Share allows $D \in \mathcal{P}$ to concurrently share a secret $S = (s^1 \ldots s^\ell)$, containing ℓ elements. On the other hand, protocol AWSS-MS-Rec-Private allows a specific party $P_\alpha \in \mathcal{P}$ to reconstruct either S or $NULL$. Similarly, protocol AWSS-MS-Rec-Public allows all the honest parties in $\mathcal{P}$ to reconstruct either S or $NULL$.

Notice that we could have executed protocol AWSS-Share ℓ times in parallel, each sharing individual elements of S. However, this will require more communication than our protocol AWSS-MS-Share for sufficiently large ℓ. Similarly, protocol AWSS-MS-Rec-Private (and AWSS-MS-Rec-Public) reconstructs all the ℓ secrets simultaneously, with a better communication complexity than individual reconstruction of ℓ secrets separately.

The Intuition: The high level idea of protocol AWSS-MS-Share is similar to AWSS-Share. For each $s^l, l = 1, \ldots, \ell$, the dealer D selects a random symmetric bivariate polynomial $F^l(x, y)$ of degree-t in x and y, where $F^l(0, 0) = s^l$ and gives his IC signature on $f_i^l(1), \ldots, f_i^l(n)$ to party P_i, for $i = 1, \ldots, n$. For this, D can execute n instances of Gen, one for each $f_i^l(j)$, for $j = 1, \ldots, n$ (this approach was used in AWSS-Share). However, this would require a total of ℓn instances of Gen (each dealing with a *single* secret) to be executed by D for every party P_i. Instead of this, a better solution would be to ask D to execute n instances of Gen, where in the j^{th} instance, D gives his IC signature *collectively* on $(f_i^1(j), f_i^2(j), \ldots, f_i^l(j))$ to party P_i.

Next each party P_i tries to *IC-commit* $(f_i^1(0), \ldots, f_i^\ell(0))$ simultaneously. For this, every pair of parties P_i and P_j privately exchange $(f_i^1(j), \ldots, f_i^\ell(j))$ and $(f_j^1(i), \ldots, f_j^\ell(i))$, along with their respective IC signature on these values. Again notice that P_i and P_j pass on their IC signature *collectively* on $(f_i^1(j), \ldots, f_i^\ell(j))$ and $(f_j^1(i), \ldots, f_j^\ell(i))$ respectively. Next the parties pair-wise check whether $f_i^l(j) = f_j^l(i)$ *for all* $l = 1, \ldots, \ell$ and if so they A-cast OK signal. After this, the remaining steps (like $WCORE$ construction, agreement on $WCORE$, etc) are same as in AWSS-Share. So essentially, the differences between AWSS-Share and AWSS-MS-Share are: (1) the way the parties give their IC signatures and (2) the conditions required for A-casting OK signal. Protocol AWSS-MS-Share is formally given in Fig. 8.3.

Remark 8.12 (D's AWSS-commitment) *We say that D is AWSS-committed to $S = (s^1, \ldots, s^\ell) \in \mathbb{F}^\ell$ if for every $l = 1, \ldots, \ell$ there is a unique degree-t polynomial $f^l(x)$ such that $f^l(0) = s^l$ and every honest P_i in $WCORE$ receives $f^l(i)$ from D and IC-commits $f^l(i)$ among the parties in OKP_i. Otherwise, we say that D is AWSS-committed to NULL. An honest D always AWSS-commits $S \in \mathbb{F}^\ell$ as in this case $f^l(x) = f_0^l(x) = F^l(x, 0)$, where $F^l(x, y)$ is the symmetric bivariate polynomial of degree-t in x and y chosen by D. But AWSS-MS-Share can not ensure that corrupted D also AWSS-commits $S \in \mathbb{F}^\ell$. This means that a corrupted D may distribute information to the parties such that, polynomial $f_0^l(x)$ defined by the $f_0^l(i) (= f_i^l(0))$ values possessed by honest P_i's in $WCORE$ may not be a degree-t polynomial for some l. In this case we say D has AWSS-committed $NULL$.*

[3] Here MS stands for multiple secrets

Figure 8.3: Sharing Phase of Protocol AWSS-MS for Sharing S Containing $\ell \geq 1$ Secrets

AWSS-MS-Share$(D, \mathcal{P}, S = (s^1 \dots s^\ell), \epsilon)$

Distribution: Code for D – Only D executes this code.

1. For $l = 1, \dots, \ell$, select a random, symmetric bivariate polynomial $F^l(x, y)$ of degree-t in x and y such that $F^l(0,0) = s^l$. Let $f_i^l(x) = F^l(x, i)$, for $l = 1, \dots, \ell$.
2. For $i = 1, \dots, n$, send $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \dots, f_i^\ell(j)))$ having $\epsilon' = \frac{\epsilon}{n^2}$ error for each $j = 1, \dots, n$ to P_i.

Verification: Code for P_i – Every party including D executes this code.

1. Wait to receive $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \dots, f_i^\ell(j)))$ having ϵ' error for $j = 1, \dots, n$ from D.
2. Check if $(f_i^l(1), \dots, f_i^l(n))$ defines degree-t polynomial for every $l = 1, \dots, \ell$. If yes then send $ICSig(P_i, P_j, \mathcal{P}, (f_i^1(j), \dots, f_i^\ell(j)))$ having ϵ' error to P_j for all $j = 1, \dots, n$.
3. If $ICSig(P_j, P_i, \mathcal{P}, (f_j^1(i), \dots, f_j^\ell(i)))$ having ϵ' error, is received from P_j and if $f_j^l(i) = f_i^l(j)$ for all $l = 1, \dots, \ell$, then A-cast $\mathtt{OK}(P_i, P_j)$.

WCORE Construction : Code for D – Only D executes this code.

1. For each P_j, build a set $OKP_j = \{P_i | D \text{ receives } \mathtt{OK}(P_i, P_j) \text{ from the A-cast of } P_i\}$. When $|OKP_j| = 2t + 1$, then P_j's *IC-commitment* on $(f_j^1(0), \dots, f_j^\ell(0))$ is over (or we may say that P_j is *IC-committed* to $(f_j^1(0), \dots, f_j^\ell(0))$) and add P_j in $WCORE$ (which is initially empty).
2. Wait until $|WCORE| = 2t + 1$. Then A-cast $WCORE$ and OKP_j for all $P_j \in WCORE$.

WCORE Verification & Agreement on WCORE : Code for P_i

1. Wait to obtain $WCORE$ and OKP_j for all $P_j \in WCORE$ from D's A-cast, such that $|WCORE| = 2t + 1$ and $|OKP_j| = 2t + 1$ for each $P_j \in WCORE$.
2. Wait to receive $\mathtt{OK}(P_k, P_j)$ for all $P_k \in OKP_j$ and $P_j \in WCORE$. After receiving all these OKs, accept the $WCORE$ and OKP_j's received from D and terminate AWSS-MS-Share.

Protocol AWSS-MS-Rec-Private and AWSS-MS-Rec-Public are straightforward extensions of protocol AWSS-Rec-Private and AWSS-MS-Rec-Public and are given in Fig. 8.4.

Since technique wise, protocol AWSS-MS is very similar to protocol AWSS, we do not provide the proofs of the properties of protocol AWSS-MS for the sake of avoiding repetition. Rather, we just state the following theorem.

Theorem 8.13 *Protocol AWSS-MS is a valid statistical AWSS scheme with $n = 3t + 1$ for ℓ secrets.*

Figure 8.4: Reconstruction Phases of AWSS-MS for Sharing S Containing ℓ Secrets

AWSS-MS-Rec-Private$(D, \mathcal{P}, S = (s^1, \ldots, s^\ell), P_\alpha, \epsilon)$
P_α**-weak-private-reconstruction of** S

Signature Revelation: Code for P_i — Every party executes this code

1. If P_i belongs to OKP_j for some $P_j \in WCORE$, then privately reveal $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ and $ICSig(P_j, P_i, \mathcal{P}, (f_j^1(i), \ldots, f_j^\ell(i)))$ to P_α, each having ϵ' error.

Local Computation: Code for P_α — Only P_α executes this code

1. For every $P_j \in WCORE$, reconstruct P_j's *IC-commitment*, say $(\overline{f_j^1}(0), \ldots, \overline{f_j^\ell}(0))$ as follows:
   (a) Construct a set $ValidP_j = \emptyset$.
   (b) Add $P_k \in OKP_j$ to $ValidP_j$ if the following conditions hold:
      i. Revelation of $ICSig(D, P_k, \mathcal{P}, (f_k^1(j), \ldots, f_k^\ell(j)))$ and $ICSig(P_j, P_k, \mathcal{P}, (f_j^1(k), \ldots, f_j^\ell(k)))$ are completed with Reveal$_\alpha$ $= (\overline{f_k^1}(j), \ldots, \overline{f_k^\ell}(j))$ and Reveal$_\alpha$ $= (\overline{f_j^1}(k), \ldots, \overline{f_j^\ell}(k))$ respectively; and
      ii. $\overline{f_k^l}(j) = \overline{f_j^l}(k)$, for $l = 1, \ldots, \ell$.
   (c) Wait until $|ValidP_j| = t + 1$. For $l = 1, \ldots, \ell$, construct a degree-t polynomial $\overline{f_j^l}(x)$ passing through the points $(k, \overline{f_j^l}(k))$ where $P_k \in ValidP_j$. For $l = 1, \ldots, \ell$, associate $\overline{f_j^l}(0)$ with $P_j \in WCORE$.
2. Wait for $\overline{f_j^1}(0), \ldots, \overline{f_j^\ell}(0)$ to be reconstructed for every P_j in $WCORE$.
3. For $l = 1, \ldots, \ell$, do the following:
   (a) Check whether the points $(j, \overline{f_j^l}(0))$ for $P_j \in WCORE$ lie on a unique degree-t polynomial $\overline{f_0^l}(x)$. If yes, then set $\overline{s^l} = \overline{f_0^l}(0)$, else set $\overline{s^l} = NULL$.
4. If $\overline{s^l} = NULL$ for any $l \in \{1, \ldots, \ell\}$, then output $\overline{S} = NULL$ and terminate AWSS-MS-Rec-Private. Else output $\overline{S} = (\overline{s^1}, \ldots, \overline{s^\ell})$ and terminate AWSS-MS-Rec-Private.

AWSS-MS-Rec-Public$(D, \mathcal{P}, S, \epsilon)$

Signature Revelation: Code for P_i — Every party executes this code

1. If P_i belongs to OKP_j for some $P_j \in WCORE$, then publicly reveal $ICSig(D, P_i, \mathcal{P}, (f_i^1(j), \ldots, f_i^\ell(j)))$ and $ICSig(P_j, P_i, \mathcal{P}, (f_j^1(i), \ldots, f_j^\ell(i)))$, each having ϵ' error.

Local Computation : Code for P_i — Every party executes this code
Same as the Local Computation of P_α in AWSS-MS-Rec-Private.

8.3.2.1 Important Notation

The following notation will be used in our AVSS protocols irrespective of which AICP is used to generate the underlying IC signatures in protocol AWSS-MS.

Notation 8.14 (Notation for Using AWSS-MS) *In our AVSS schemes,*

- *We will invoke AWSS-MS-Share as AWSS-MS-Share*$(D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x)), \epsilon)$ *where D is asked to choose symmetric bivariate polynomials $F^1(x,y), \ldots, F^\ell(x,y)$ each of degree-t in x and y such that $F^l(x,0) = f^l(x)$ holds for $l = 1, \ldots, \ell$. D then tries to give $F^l(x,i)$ and hence $F^l(0,i) = f^l(i)$ to party P_i, for $l = 1, \ldots, \ell$.*

- *Similarly, AWSS-MS-Rec-Private will be invoked as AWSS-MS-Rec-Private*$(D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x)), P_\alpha, \epsilon)$ *to enable the P_α-weak-private-reconstruction of $(f^1(x), \ldots, f^\ell(x))$.*

- *Similarly, AWSS-MS-Rec-Public will be invoked as AWSS-MS-Rec-Public*$(D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x)), \epsilon)$ *to enable public reconstruction of $(f^1(x), \ldots, f^\ell(x))$ or $NULL$.*

8.3.3 Deriving Two AWSS Protocols for Single Secret from Protocol AWSS

We now derive two AWSS protocols with different communication complexity out of protocol AWSS by substituting AICPs MVMS-AICP-I and MVMS-AICP-II. As mentioned earlier, the usage of AICP will decide the field over which the derived AWSS will work.

8.3.3.1 AWSS Protocol with MVMS-AICP-I as Building Block

In protocol AWSS, if the used IC signatures are generated using protocol MVMS-ICP-I for $\ell = 1$, then we obtain a AWSS protocol which we denote by AWSS-I. Furthermore, the sub-protocols are denoted by (AWSS-I-Share, AWSS-I-Rec-Private, AWSS-I-Rec-Public). Our protocol AWSS-I involves an error probability of ϵ.

To bound the error probability by ϵ, the computation in AWSS-I is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^3\kappa 2^{-\kappa}$. This is derived from the fact that in AWSS-I, MVMS-AICP-I is invoked with $\frac{\epsilon}{n^2}$ error probability and as mentioned in section 7.3 of Chapter 7, $\epsilon \geq n\kappa 2^{-\kappa}$ should hold to bound error probability of MVMS-AICP-I by ϵ. So here each element from the field is represented by $\kappa = \log|\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits (this can be derived using $n = \mathcal{O}(\log \frac{1}{\epsilon})$).

We now present the communication complexity of protocol AWSS-I.

Lemma 8.15 (Communication Complexity of AWSS-I)

- *Protocol AWSS-I-Share incurs a private communication of $\mathcal{O}(n^3(\log \frac{1}{\epsilon})^2)$ bits and A-cast of $\mathcal{O}(n^2 \log n)$ bits.*

- *Protocol AWSS-I-Rec-Private privately communicates $\mathcal{O}(n^3(\log \frac{1}{\epsilon})^2)$ bits.*

- *Protocol AWSS-I-Rec-Public involves A-cast of $\mathcal{O}(n^3(\log \frac{1}{\epsilon})^2)$ bits.*

Proof: In AWSS-I-Share, there are $\mathcal{O}(n^2)$ instances of Gen and Ver (of MVMS-AICP-I), each dealing with one value (substituting $\ell = 1$) and executed with an error probability of $\epsilon' = \frac{\epsilon}{n^2}$. From Theorem 7.8, this requires a private communication of $\mathcal{O}(n^3(\log \frac{n^2}{\epsilon})^2) = \mathcal{O}(n^3(\log \frac{1}{\epsilon})^2)$ bits, as $n = \mathcal{O}(\log \frac{1}{\epsilon})$. Moreover, there are A-cast of $\mathcal{O}(n^2 \log n)$ bits for OK signals (identity of each party can be expressed by $\log n$ bits and an OK signal contains identity of two parties). In addition, there is A-cast of $WCORE$ containing the identity of $2t+1$ parties and OK sets corresponding to each party in $WCORE$, where each OK set contains the identity of $2t+1$ parties. Now the identity of a party can be represented by $\mathcal{O}(\log n)$ bits. So in total, AWSS-I-Share incurs a private communication of $\mathcal{O}(n^3(\log \frac{1}{\epsilon})^2)$ bits and A-cast of $\mathcal{O}(n^2 \log n)$ bits.

In AWSS-I-Rec-Private, there are $\mathcal{O}(n^2)$ instances of Reveal-Private of our MVMS-AICP-I, each dealing with $\ell = 1$ value. This requires a private communication of $\mathcal{O}(n^3(\log \frac{1}{\epsilon})^2)$ bits. Similarly, the communication complexity of AWSS-I-Rec-Public follows from Theorem 7.8 and the fact that there are $\mathcal{O}(n^2)$ instances of Reveal-Public of our MVMS-AICP-I. □

8.3.3.2 AWSS Protocol with MVMS-AICP-II as Building Block

In protocol AWSS, if the used IC signatures are generated using protocol MVMS-ICP-II for $\ell = 1$, then we obtain a AWSS protocol which we denote by AWSS-II. Furthermore, the sub-protocols are denoted by (AWSS-II-Share, AWSS-II-Rec-Private, AWSS-II-Rec-Public). Our protocol AWSS-II involves an error probability of ϵ.

To bound the error probability by ϵ, the computation in AWSS-II is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^3 2^{-\kappa}$. This is derived from the fact that in AWSS-II, MVMS-AICP-II is invoked with $\frac{\epsilon}{n^2}$ error probability and as mentioned in section 7.4 of Chapter 7, $\epsilon \geq n2^{-\kappa}$ should hold to bound error probability of MVMS-AICP-II by ϵ. So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits (this can be derived using $n = \mathcal{O}(\log \frac{1}{\epsilon})$).

We now present the communication complexity of protocol AWSS-II.

Lemma 8.16 (Communication Complexity of AWSS-II)

- *Protocol AWSS-II-Share incurs a private communication of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits.*
- *Protocol AWSS-II-Rec-Private privately communicates $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits.*
- *Protocol AWSS-II-Rec-Public involves A-cast of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits.*

PROOF: Follows from Theorem 7.16 and the proof of communication complexity of AWSS-I. □

8.3.4 Deriving Two AWSS Protocols for Multiple Secrets from Protocol AWSS-MS

8.3.4.1 AWSS Protocol with MVMS-AICP-I as Building Block

In protocol AWSS-MS, if the used IC signatures are generated using protocol MVMS-ICP-I, then we obtain a AWSS protocol which we denote by AWSS-MS-I. Furthermore, the sub-protocols are denoted by (AWSS-MS-I-Share, AWSS-MS-I-Rec-Private, AWSS-MS-I-Rec-Public). Our protocol AWSS-MS-I involves an error probability of ϵ.

To bound the error probability by ϵ, the computation in AWSS-MS-I is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^3\kappa 2^{-\kappa}$. This is derived in the same way as done for AWSS-I.

We now present the communication complexity of protocol AWSS-MS-I.

Lemma 8.17 (Communication Complexity of AWSS-MS-I)

- *Protocol AWSS-MS-I-Share privately communicates $\mathcal{O}((\ell n^2 + n^3 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-casts $\mathcal{O}(n^2 \log n)$ bits.*
- *Protocol AWSS-MS-I-Rec-Private privately communicates $\mathcal{O}((\ell n^2 + n^3 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits.*
- *Protocol AWSS-MS-I-Rec-Public involves A-cast of $\mathcal{O}((\ell n^2 + n^3 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits.*

PROOF: In AWSS-MS-I-Share, n^2 instances of MVMS-AICP-I are executed. In addition, there are n^2 A-cast of OK(*,*) signals. This will require A-cast of $\mathcal{O}(n^2 \log n)$ bits. So AWSS-MS-I-Share involves a private communication of $\mathcal{O}((\ell n^2 + n^3 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-casts of $\mathcal{O}(n^2 \log n)$ bits.

AWSS-MS-I-Rec-Private executes $\mathcal{O}(n^2)$ instances of Reveal-Private of our MVMS-AICP-I. This requires a private communication of $\mathcal{O}((\ell n^2 + n^3 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits. Similarly, the communication complexity of AWSS-MS-I-Rec-Public follows from Theorem 7.8 and the fact that it executes $\mathcal{O}(n^2)$ instances of Reveal-Public of our MVMS-AICP-I. □

Now comparing the communication complexity of AWSS-MS-I and AWSS-I, we find that AWSS-MS-I provides better communication complexity that ℓ parallel execution of AWSS-I for ℓ individual secrets.

8.3.4.2 AWSS Protocol with MVMS-AICP-II as Building Block

In protocol AWSS-MS, if the used IC signatures are generated using protocol MVMS-ICP-II, then we obtain a AWSS protocol which we denote by AWSS-MS-II. Furthermore, the sub-protocols are denoted by (AWSS-MS-II-Share, AWSS-MS-II-Rec-Private, AWSS-MS-II-Rec-Public). Our protocol AWSS-MS-II involves an error probability of ϵ.

To bound the error probability by ϵ, the computation in AWSS-MS-II is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^3 2^{-\kappa}$. This is derived in the same way as done for AWSS-II.

We now present the communication complexity of protocol AWSS-MS-II.

Lemma 8.18 (Communication Complexity of AWSS-MS-II)

- *Protocol AWSS-MS-II-Share incurs a private communication of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits.*
- *Protocol AWSS-MS-II-Rec-Private privately communicates $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits.*
- *Protocol AWSS-MS-II-Rec-Public involves A-cast of $\mathcal{O}((\ell n^2 + n^3) \log \frac{1}{\epsilon})$ bits.*

PROOF: Follows from Theorem 7.16 and the proof of communication complexity of AWSS-MS-I. □

Now comparing the communication complexity of AWSS-MS-II and AWSS-II, we find that AWSS-MS-II provides better communication complexity that ℓ parallel execution of AWSS-II for ℓ individual secrets.

8.4 Our Weak Statistical AVSS protocol

For the sake of simplicity, we first present our AVSS protocol sharing a single secret and then extend the protocol for multiple (i.e ℓ) secrets. We will later show that dealing with multiple secrets concurrently in our protocol provides with better communication complexity that multiple executions of protocol dealing with single secret. We may use any one of the AWSS presented in the previous section as a building block for our protocol. But we will use AWSS-II and corresponding multiple secret AWSS AWSS-MS-II for our single and multiple secret version of AVSS respectively. We dedicate a subsection at the end of this section to state the reason for our choice. In the sequel, our AVSS protocols are described without hinting on which AWSS is used, as the AWSS can be replaced by either one of the two AWSS protocols described in the previous section. Lastly, our weak statistical AVSS protocol is much simpler than our strong statistical AVSS presented in the next section (it will be evident at the end of this chapter).

8.4.1 Our Weak Statistical AVSS Scheme for Sharing a Single Secret

In this section, we present our novel AVSS scheme called WAVSS[4] consisting of sub-protocols (WAVSS-Share, WAVSS-Rec-Private, WAVSS-Rec-Public). While WAVSS-Share allows D to share a secret s, WAVSS-Rec-Public enables public reconstruction of D's shared secret and WAVSS-Rec-Private enables private reconstruction of D's shared secret by some specific party say $P_\alpha \in \mathcal{P}$. Moreover, if D is *corrupted*, then s can be either from $\mathbb{F}$ or it can be $NULL$ (in a sense explained in the sequel).

High Level Idea of WAVSS-Share: To design WAVSS-Share, we use the general approach of [138, 137, 125, 73, 109] used in synchronous settings for designing VSS using WSS as a black box. The high level idea of WAVSS-Share is as follows: D selects a symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y, such that $F(0, 0) = s$ and sends $f_i(x) = F(x, i)$ to party P_i. Now the parties communicate with each other to perform what we say *commitment upon verification*. Here each party P_i is asked to *commit* the polynomial $f_i(x)$, that he has received from D. However, P_i is allowed to commit $f_i(x)$, only after the parties have *verified* that they have received same points on $f_i(x)$ from D as well as P_i. More formally, to achieve *commitment upon verification*, party P_i, acting as a dealer, shares his polynomial $f_i(x)$ by initiating an instance of AWSS-Share (see Notation 8.11 for the meaning of sharing polynomial using AWSS-Share). Since party P_j receives $f_i(j)$ from P_i as part of AWSS-Share, he can check whether $f_i(j) \stackrel{?}{=} f_j(i)$, as ideally $f_i(j) = f_j(i)$ should hold in case of honest D, P_i and P_j. A party P_j participates in the remaining steps of the instance of AWSS-Share where P_i is the dealer, only if

[4] WAVSS stands for Weak statistical AVSS

$f_i(j) = f_j(i)$ holds. Once *commitment upon verification* is over, the parties want to agree on a set of at least $2t+1$ parties, denoted as $VCORE$, such that for every P_j in $VCORE$, P_j's instance of AWSS-Share terminates with a $WCORE$ set, denoted as $WCORE^{P_j}$ and $|VCORE \cap WCORE^{P_j}| \geq 2t+1$. Informally, this means that each party $P_j \in VCORE$ has 'successfully' committed his polynomial $f_j(x)$ to at least $2t+1$ parties in $VCORE$, who have verified that they have received correct points on $f_j(x)$. We will refer this commitment as P_j's *AWSS-commitment* on $f_j(x)$. It should be noted that *AWSS-Commitment* is strictly stronger commitment than *IC-commitment* that was enforced in AWSS-Share. These two commitments can be distinguished by the facts that when both D and P_j are corrupted (a) *AWSS-commitment* ensures that reconstruction of *AWSS-commitment* can not be changed to some other value other than $NULL$ and (b) *IC-commitment* can not ensure the same. The agreement on $VCORE$ and corresponding $WCORE^{P_j}$'s is achieved using a mechanism, similar to the one used in AWSS-Share for achieving agreement on $WCORE$ and corresponding OK sets. Protocol WAVSS-Share is formally presented in Fig. 8.5.

Remark 8.19 (D's AVSS-commitment) *We say that D has AVSS-committed $s \in \mathbb{F}$ in WAVSS-Share if there is a unique symmetric bivariate polynomial $F(x,y)$ of degree-t in x and y, such that $F(0,0) = s$ and every honest P_i in $VCORE$ receives $f_i(x) = F(x,i)$ from D and AWSS-commits $f_i(0)$ using $f_i(x)$ among the parties in $WCORE^{P_i}$. Otherwise, we say that D has committed $NULL$. Notice that the above condition implies that there exist a unique degree-t univariate polynomial $f(x)(= f_0(x) = F(x,0))$ such that $f(0) = s$ and every honest $P_i \in VCORE$ receives $f(i)(= f_0(i) = f_i(0))$ from D.* **The value $f(i)$ is referred as i^{th} share of** *s. An honest D always AVSS-commits s from $\mathbb{F}$ as he always chooses a proper symmetric bivariate polynomial $F(x,y)$ and properly distributes $f_i(x) = F(x,i)$ to party P_i. But AVSS-Share can not ensure that corrupted D also commits $s \in \mathbb{F}$. When a corrupted D commits $NULL$, the $f_i(x)$ polynomials of the honest parties in $VCORE$ do not define a symmetric bivariate polynomial of degree-t in x and y. This further implies that there will be an honest pair (P_γ, P_δ) in $VCORE$ such that $f_\gamma(\delta) \neq f_\delta(\gamma)$. When $s = NULL$, we say that D's AVSS-committed secret s is not meaningful.*

In our following discussion, we show that irrespective of whether s is chosen from $\mathbb{F}$ or it is $NULL$, s will be reconstructed in reconstruction phase, except with probability ϵ.

High Level Idea of WAVSS-Rec-Public & WAVSS-Rec-Private: In WAVSS-Rec-Public, D's AVSS-committed secret is recovered with the help of the parties in $VCORE$ and $WCORE^{P_j}$'s. Specifically, in the reconstruction phase, for every $P_j \in VCORE$, *AWSS-commitment* on $f_j(x)$ is revealed by reconstructing it with the help of the parties in $WCORE^{P_j}$. This is done by executing an instance of AWSS-Rec with the parties in $WCORE^{P_j}$. This results in the reconstruction of either $f_j(x)$ or NULL depending on whether P_j is honest or corrupted. Since $|VCORE| \geq 2t+1$, for (at least $t+1$) honest parties, $f_j(x)$'s will be recovered correctly. Now with the $f_j(x)$'s, $F(x,y)$ will be reconstructed. The formal details of WAVSS-Rec-Public and WAVSS-Rec-Private are given in Fig. 8.6.

We now prove the properties of protocol WAVSS assuming WAVSS-Rec-Public as the reconstruction phase protocol. The proofs can be twisted little bit for the case when WAVSS-Rec-Private is assumed as the reconstruction phase protocol.

Figure 8.5: Sharing Phase of our Weak Statistical AVSS Scheme for Sharing a Single Secret s with $n = 3t+1$

WAVSS-Share($D, \mathcal{P}, s, \epsilon$)

Distribution: Code for D — Only D executes this code

1. Select a random symmetric bivariate polynomial $F(x, y)$ of degree-t in x and y such that $F(0, 0) = s$ and send $f_i(x) = F(x, i)$ to party P_i, for $i = 1, \ldots, n$.

Commitment upon Verification: Code for P_i — Every party, including D executes this code

1. Wait to obtain $f_i(x)$ from D.
2. If $f_i(x)$ is a degree-t polynomial then invoke AWSS-Share($P_i, \mathcal{P}, f_i(x), \epsilon'$) (See Notation 8.11 for the syntax) with $\epsilon' = \frac{\epsilon}{n}$. We call this instance of AWSS-Share initiated by P_i as AWSS-Share$_i$.
3. As a part of the execution of AWSS-Share$_j$, wait to receive $f_j(i)$ from P_j. Then check $f_i(j) \stackrel{?}{=} f_j(i)$. If the test passes then participate in AWSS-Share$_j$ and act according to the remaining steps of AWSS-Share$_j$.

VCORE Construction: Code for D — Only D executes this code

1. If AWSS-Share$_j$ is terminated, then denote corresponding $WCORE$ and OKP_k sets by $WCORE^{P_j}$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$. Add P_j in a set $VCORE$ (initially empty).
2. Keep updating $VCORE$, $WCORE^{P_j}$ and corresponding $OKP_k^{P_j}$'s for every $P_j \in VCORE$ upon receiving new A-casts of the form OK(.,.) (during AWSS-Share$_j$'s), until for at least $2t+1$ $P_j \in VCORE$, the condition $|VCORE \cap WCORE^{P_j}| \geq 2t+1$ is satisfied. Remove (from $VCORE$) all $P_j \in VCORE$ for whom the above condition is not satisfied.
3. A-cast $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$. Conclude that each $P_j \in VCORE$ is *AWSS-committed* to $f_j(x)$.

VCORE Verification & Agreement on VCORE: Code for P_i — Every party executes this code

1. Wait to receive $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ from D's A-cast.
2. Wait to terminate AWSS-Share$_j$ corresponding to every P_j in $VCORE$.
3. Wait to receive OK(P_m, P_k) for every $P_k \in WCORE^{P_j}$ and every $P_m \in OKP_k^{P_j}$, corresponding to every $P_j \in VCORE$.
4. Accept $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ and terminate WAVSS-Share.

Figure 8.6: Reconstruction Phase of our Weak Statistical AVSS Scheme for Sharing a Single Secret s with $n = 3t + 1$

WAVSS-Rec-Private$(D, \mathcal{P}, s, P_\alpha \epsilon)$

P_α-weak-private-reconstruction of $f_j(x)$ for every $P_j \in VCORE$: (Code for P_i)

1. Participate in AWSS-Rec-Private$(P_j, \mathcal{P}, f_j(x), P_\alpha, \epsilon')$ for every $P_j \in VCORE$ with $WCORE^{P_j}$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$, where $\epsilon' = \frac{\epsilon}{n}$. We denote AWSS-Rec-Private$(P_j, \mathcal{P}, f_j(x), P_\alpha, \epsilon')$ by AWSS-Rec-Private$_j$.

Local Computation: Code for P_α

1. For every $P_j \in VCORE$, obtain either $\overline{f_j}(x)$ or $NULL$ from P_α-weak-private-reconstruction. Add $P_j \in VCORE$ to REC if $\overline{f_j}(x)$ is obtained.
2. Wait until $|REC| = t+1$. For every pair $(P_\gamma, P_\delta) \in REC$ check $\overline{f_\gamma}(\delta) \stackrel{?}{=} \overline{f_\delta}(\gamma)$. If the test passes for every pair of parties then recover $\overline{F}(x, y)$ using $\overline{f_j}(x)$'s corresponding to each $P_j \in REC$ and reconstruct $\overline{s} = \overline{F}(0, 0)$. Else reconstruct $\overline{s} = NULL$. Finally output $\overline{s}$ and terminate WAVSS-Rec-Private.

WAVSS-Rec-Public$(D, \mathcal{P}, s, \epsilon)$

Public reconstruction of $f_j(x)$ for every $P_j \in VCORE$: (Code for P_i)

1. Participate in AWSS-Rec-Public$(P_j, \mathcal{P}, f_j(x), \epsilon')$ for every $P_j \in VCORE$ with $WCORE^{P_j}$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$, where $\epsilon' = \frac{\epsilon}{n}$. We denote AWSS-Rec-Public$(P_j, \mathcal{P}, f_j(x), \epsilon')$ by AWSS-Rec-Public$_j$.

Local Computation: Code for P_i

For every $P_j \in VCORE$, obtain either $\overline{f_j}(x)$ or $NULL$ from public reconstruction. Remaining code is same as the code for P_α presented above for WAVSS-Rec-Private

Lemma 8.20 (AVSS-Termination) *Protocols WAVSS satisfies* **termination** *property of Definition 8.3.*

PROOF:

- **Termination 1:** Notice that in WAVSS-Share, D keeps on adding new parties to $WCORE^{P_j}$ in instance AWSS-Share$_j$, even after $WCORE^{P_j}$ contains $2t + 1$ parties. So if D is honest, then corresponding to every honest P_j, $2t + 1$ honest parties will be eventually included in $WCORE^{P_j}$. Now eventually at least $2t + 1$ honest parties will be included in $VCORE$, such that $|VCORE \cap WCORE^{P_j}| \geq 2t+1$ for each $P_j \in VCORE$. Now from similar argument given in **Termination 1** of Lemma 8.7, all honest parties will eventually agree on $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ and will terminate WAVSS-Share.

- **Termination 2:** If some honest party has terminated WAVSS-Share then it implies that he has received $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ from the A-cast of D and checked their validity. So by the property of A-cast, every other honest party will also eventually do the same and terminate WAVSS-Share.

- **Termination 3:** Follows from the fact that corresponding to each honest $P_j \in VCORE$, every honest P_i will eventually terminate AWSS-Rec-Public$_j$ (from **Termination 3** of Lemma 8.7), except with an error probability of ϵ'. As there are at least $t+1$ honest parties in $VCORE$, AWSS-Rec-Public corresponding to all the honest parties will terminate with probability at least $(1-(t+1)\epsilon') \approx (1-\epsilon)$. □

Lemma 8.21 (AVSS-Correctness) *Protocol WAVSS satisfies* **correctness** *property of Definition 8.3.*

PROOF:

- **Correctness 1:** We have to consider the case when D is honest. If D is *honest* then we prove that except with probability ϵ', for every $P_i \in REC$, P_i's *AWSS-Commitment* will be reconstructed correctly. In other words, AWSS-Rec-Public$_i$ will reconstruct $\overline{f_i}(x)$ which is same as $f_i(x)$ selected by honest D. For every *honest* $P_i \in REC$ this is trivially true and follows from the **Correctness1** of our AWSS scheme. We have to prove the above statement for a corrupted $P_i \in REC$. If a corrupted P_i belongs to REC, it implies that AWSS-Rec-Public$_i$ is successful (i.e., the output is non-NULL) and AWSS-Share$_i$ had terminated during WAVSS-Share, such that $|VCORE \cap WCORE^{P_i}| \geq 2t+1$. The above statements have the following implications:

  > P_i must have agreed with the honest parties of $WCORE^{P_i}$ with respect to the common values given by D. This means that as a part of AWSS-Share$_i$, P_i handed over $f_j(i)$ to an honest P_j (in $WCORE^{P_i}$).

  The above further implies that P_i must have committed (to the honest parties in $WCORE^{P_i}$ whose count is at least $t+1$) $f_i(x)$. Thus if AWSS-Rec-Public$_i$ is successful, then except with probability ϵ', $\overline{f_i}(x) = f_i(x)$. Since D is honest, $\overline{f_i}(x)$'s corresponding to $P_i \in REC$ will define $\overline{F}(x,y) = F(x,y)$. In the worst case, there can be at most t corrupted parties in REC and hence except with probability $\epsilon' t \approx \epsilon$, $\overline{f_i}(x)$'s corresponding to each $P_i \in REC$ will define $\overline{F}(x,y) = F(x,y)$ and thus $s = \overline{F}(0,0) = F(0,0)$ will be recovered.

- **Correctness 2:** Here we have to consider the case when D is corrupted. Now there are two cases: (a) D's *AVSS-committed* secret s belongs to $\mathbb{F}$; (b) D's *AVSS-committed* secret s is $NULL$. Whatever may be case, we show that except with probability ϵ, each honest party will reconstruct s.

  1. Let $s = NULL$. Now for every honest $P_i \in VCORE$, AWSS-Rec-Public$_i$ will reconstruct $\overline{f_i}(x)$ correctly and thus P_i will be added to REC, except with error probability ϵ'. Consequently since there are at least $t+1$ honest parties in $VCORE$, all the honest parties from $VCORE$

will be added to REC except with error probability of $n\epsilon' = \epsilon$. Now irrespective of the remaining (corrupted) parties included in REC, the consistency checking (i.e., $\overline{f_\gamma}(\delta) \stackrel{?}{=} \overline{f_\delta}(\gamma)$) will fail for some pair (P_γ, P_δ) of honest parties and thus $NULL$ will be reconstructed.

2. On the other hand, let $s \in \mathbb{F}$ (i.e *meaningful*) and $s = F(0,0)$. This means that $F(x,y)$ is defined by the $f_i(x)$'s of the honest parties in $VCORE$. This case now completely resembles with the case when D is honest and hence the proof follows from the proof of **Correctness 1** (presented above). □

Lemma 8.22 (AVSS-Secrecy) *Protocol WAVSS-Share satisfies secrecy property of Definition 8.3.*

PROOF: We have to consider the case when D is honest. Without loss of generality, let $P_1, \ldots, P_t$ be under the control of $\mathcal{A}_t$. It is easy to see that through out WAVSS-Share, $\mathcal{A}_t$ will know $f_1(x), \ldots, f_t(x)$ and t points on $f_{t+1}(x), \ldots, f_n(x)$. However, from the property of symmetric polynomial of degree-t in x and y [46], the adversary $\mathcal{A}_t$ will lack one more point on $F(x,y)$ to uniquely interpolate $F(x,y)$. Hence $s = F(0,0)$ will be information theoretically secure. □

Theorem 8.23 *Protocol WAVSS is a valid weak statistical AVSS scheme for a single secret.*

PROOF: The proof follows from Lemma 8.20, Lemma 8.21 and Lemma 8.22. □

Remark 8.24 *Protocol WAVSS-Share does not force corrupted D to AVSS-commit some meaningful secret (i.e., an element from $\mathbb{F}$). Hence, the secret s, AVSS-committed by a corrupted D can be either from $\mathbb{F}$ or $NULL$. We may assume that if D's AVSS-committed secret is $NULL$, then D has AVSS-committed some predefined value $s^* \in \mathbb{F}$, which is known publicly. Hence in WAVSS-Rec-Public, whenever $NULL$ is reconstructed, every honest party replaces $NULL$ by the predefined secret s^*. Interpreting this way, we say that our AVSS scheme allows D to AVSS-commit secret from $\mathbb{F}$.*

8.4.2 Deciding The Choice of AWSS Protocol

For our ABA protocol presented in Chapter 9, it is sufficient to design a weak statistical AVSS with *public reconstruction*. Now let us analyze the communication complexity of our WAVSS protocol by substituting AWSS-I and AWSS-II.

Communication complexity of WAVSS using AWSS-I as a Black box: Protocol WAVSS-Share incurs a private communication of $\mathcal{O}(n^4(\log\frac{1}{\epsilon})^2)$ bits and A-cast of $\mathcal{O}(n^3 \log n)$ bits (as WAVSS-Share invokes at most n instances of AWSS-Share). Protocol WAVSS-Rec-Private incurs private communication of $\mathcal{O}(n^4(\log\frac{1}{\epsilon})^2)$ bits (as WAVSS-Rec-Private invokes at most n instances of AWSS-Rec-Private). Protocol WAVSS-Rec-Public incurs A-cast communication of $\mathcal{O}(n^4(\log\frac{1}{\epsilon})^2)$ bits (as WAVSS-Rec-Public invokes at most n instances of AWSS-Rec-Public).

Communication complexity of WAVSS using AWSS-II as a Black box: Protocol WAVSS-Share incurs a private communication of $\mathcal{O}(n^4 \log\frac{1}{\epsilon})$ bits and

A-cast of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits. Protocol WAVSS-Rec-Private incurs private communication of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits. Protocol WAVSS-Rec-Public incurs A-cast communication of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits.

So if we consider WAVSS with *public reconstruction* i.e (WAVSS-Share,WAVSS-Rec-Public) then the total communication is better when AWSS-II is used as black box (which is private communication and A-cast communication of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits). So we will consider AWSS-II as a black box for WAVSS and state the communication complexity of WAVSS in the following theorem. Before that we fix the field $\mathbb{F}$ over which WAVSS should work to bound the error probability by ϵ.

To bound the error probability by ϵ, the computation in WAVSS is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^4 2^{-\kappa}$. This is derived from the fact that in WAVSS, AWSS-II is invoked with $\frac{\epsilon}{n}$ error probability and as mentioned in subsection 8.4.1, $\epsilon \geq n^3 2^{-\kappa}$ should hold to bound error probability of AWSS-II by ϵ. So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

Theorem 8.25 (Communication Complexity of WAVSS) *Using AWSS-II as building block, the communication complexity of protocol WAVSS becomes as follows*

- *Protocol WAVSS-Share incurs a private communication of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits.*
- *Protocol WAVSS-Rec-Public incurs A-cast of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits.*

PROOF: The proof follows from Lemma 8.16 and the fact that in WAVSS-Share, there can be $\Theta(n)$ instances of AWSS-Share, each executed with an error probability of $\epsilon' = \frac{\epsilon}{n}$. Moreover, in WAVSS-Rec-Public there can be $\Theta(n)$ instances of AWSS-Rec-Public. □

8.4.3 Our Weak Statistical AVSS Scheme for Sharing Multiple Secrets

We now extend protocol WAVSS to WAVSS-MS[5] consisting of sub-protocols (WAVSS-MS-Share, WAVSS-MS-Rec-Private, WAVSS-MS-Rec-Pubic). Protocol WAVSS-MS-Share allows $D \in \mathcal{P}$ to concurrently share a secret $S = (s^1 \ldots s^\ell)$, containing ℓ elements. Moreover, if D is corrupted then either $S \in \mathbb{F}^\ell$, where each element of S belongs to $\mathbb{F}$ or $S = NULL$ (in a sense explained in the sequel). Protocol WAVSS-MS-Rec-Public allows the parties in $\mathcal{P}$ to reconstruct S. Protocol WAVSS-MS-Rec-Private allows a specific party in $P_\alpha \in \mathcal{P}$ to reconstruct S.

A simple approach for sharing ℓ secrets would be to execute WAVSS-Share ℓ times in parallel, each sharing a single secret. From Theorem 8.25, this naive approach would require a private communication and A-cast of $\mathcal{O}(\ell n^4 \log \frac{1}{\epsilon})$ bits. On the other hand, protocol WAVSS-MS-Share shares all elements of S concurrently, requiring a private communication and A-cast of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits. Thus for sufficiently large ℓ, the communication complexity of WAVSS-MS-Share is less than what would have been required by ℓ parallel executions of WAVSS-Share. Similarly, protocol WAVSS-MS-Rec-Public reconstructs all the ℓ secrets

[5] Here MS stands for multiple secrets

simultaneously, incurring A-cast communication of $\mathcal{O}((\ell n^3 + n^4)\log\frac{1}{\epsilon})$ bits which is much better than ℓ parallel execution of WAVSS-Rec-Public for single secrets.

The Intuition: The high level idea of WAVSS-MS-Share is similar to WAVSS-Share. Specifically, for each $s^l \in S$, the dealer D selects a symmetric bivariate polynomial $F^l(x,y)$ of degree-t in x and y, such that $F^l(0,0) = s^l$ and sends $f_i^l(x) = F^l(x,i)$ to party P_i. Then each party P_i is asked to *AWSS-commit* his received polynomials $f_i^1(x), \ldots, f_i^\ell(x)$. However, instead of executing ℓ instances of AWSS-Share, one for committing each $f_i^l(x)$, party P_i executes a *single* instance of AWSS-MS-Share to commit $f_i^1(x), \ldots, f_i^\ell(x)$ simultaneously. It is this step, which leads to the reduction in the communication complexity of WAVSS-MS-Share. The remaining steps like $VCORE$ construction, agreement on $VCORE$, etc are similar to protocol WAVSS-Share. Protocol WAVSS-MS-Share is formally presented in Fig. 8.7.

Remark 8.26 (D's AVSS-commitment) *We say that D has AVSS-committed $S = (s^1, \ldots, s^\ell) \in \mathbb{F}^\ell$ in WAVSS-MS-Share if for every $l = 1, \ldots, \ell$ there is a unique degree-t symmetric bivariate polynomial $F^l(x,y)$ such that $F^l(0,0) = s^l$ and every honest P_i in $VCORE$ receives $f_i^l(x) = F^l(x,i)$ from D and AWSS-commits $f_i^l(0)$ using $f_i^l(x)$ among the parties in $WCORE^{P_i}$. Otherwise, we say that D has committed $NULL$. Notice that the above condition implies that for $l = 1, \ldots, \ell$ there exist a unique degree-t univariate polynomial $f^l(x)(= f_0^l(x) = F^l(x,0))$ such that $f^l(0) = s^l$ and every honest $P_i \in VCORE$ receives $f^l(i)(= f_0^l(i) = f_i^l(0))$ from D.* **The value $f^l(i)$ is referred as i^{th} share of s^l.** *An honest D always commits s^l from $\mathbb{F}$ as he always chooses a proper symmetric bivariate polynomial $F^l(x,y)$ and properly distributes $f_i^l(x) = F^l(x,i)$ to party P_i. But WAVSS-MS-Share can not ensure that corrupted D also commits $s^l \in \mathbb{F}$ for all l. When a corrupted D commits $NULL$, the $f_i^l(x)$ polynomials of the honest parties in $VCORE$ do not define a symmetric bivariate polynomial of degree-t in x and y for at least one l. This further implies that there will be an honest pair (P_γ, P_δ) in $VCORE$ such that $f_\gamma^l(\delta) \neq f_\delta^l(\gamma)$. If S belongs to $\mathbb{F}^\ell$, then it is considered as meaningful.*

Protocol WAVSS-MS-Rec-Private and WAVSS-MS-Rec-Public are straightforward extension of protocol WAVSS-Rec-Private and WAVSS-Rec-Public respectively and they appear in Fig. 8.8.

We do not provide the proof of the properties of protocol WAVSS-MS, as it will be the repetition of the proofs provided for protocol WAVSS. For the sake of completeness, we state the following theorem.

Theorem 8.27 *Protocol WAVSS-MS is a valid weak statistical AVSS scheme for multiple secrets.*

Remark 8.28 *As mentioned earlier, Protocol WAVSS-MS-Share does not force corrupted D to AVSS-commit some meaningful secret (i.e., S, containing ℓ elements from $\mathbb{F}$). We may assume that if D's AVSS-committed secret is $NULL$, then D has AVSS-committed some predefined $S^* \in \mathbb{F}^\ell$, which is known publicly. Hence in WAVSS-MS-Rec, whenever $NULL$ is reconstructed, every honest party replaces $NULL$ by the predefined S^*. Interpreting this way, we say that our AVSS scheme allows D to AVSS-commit secret from $\mathbb{F}^\ell$.*

Figure 8.7: Sharing Phase of Weak Statistical AVSS Scheme for Sharing a Secret S Containing ℓ Elements

WAVSS-MS-Share$(D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \epsilon)$

Distribution: Code for D — Only D executes this code.

1. For $l = 1, \ldots, \ell$, select a random symmetric bivariate polynomial $F^l(x, y)$ of degree-t in x and y such that $F^l(0, 0) = s^l$ and send $f_i^l(x) = F^l(x, i)$ to party P_i, for $i = 1, \ldots, n$.

Commitment upon Verification: Code for P_i — Every party in $\mathcal{P}$, including D, executes this code.

1. Wait to obtain $f_i^1(x), \ldots, f_i^\ell(x)$ from D.
2. If $f_i^1(x), \ldots, f_i^\ell(x)$ are degree-t polynomials then as a dealer, execute AWSS-MS-Share$(P_i, \mathcal{P}, (f_i^1(x), \ldots, f_i^\ell(x)), \epsilon')$ (see Notation 8.14 for the syntax) such that $\epsilon' = \frac{\epsilon}{n}$. We call this instance of AWSS-MS-Share initiated by P_i as AWSS-MS-Share$_i$.
3. As a part of the execution of AWSS-MS-Share$_j$, wait to receive $f_j^l(i)$, for $l = 1, \ldots, \ell$ from P_j. Then check $f_i^l(j) \stackrel{?}{=} f_j^l(i)$. If the test passes for all $l = 1, \ldots, \ell$ then participate in AWSS-MS-Share$_j$ and act according to the remaining steps of AWSS-MS-Share$_j$.

VCORE Construction: Code for D – Only D executes this code

1. If AWSS-MS-Share$_j$ is terminated, then denote corresponding $WCORE$ and OKP_k sets by $WCORE^{P_j}$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$. Add P_j in a set $VCORE$ (initially empty).
2. Keep updating $VCORE$, $WCORE^{P_j}$ and corresponding $OKP_k^{P_j}$'s for every $P_j \in VCORE$ upon receiving new A-casts of the form $\texttt{OK}(.,.)$ (during AWSS-MS-Share$_j$'s), until for at least $2t+1$ $P_j \in VCORE$, the condition $|VCORE \cap WCORE^{P_j}| \geq 2t+1$ is satisfied. Exclude every other $P_j \in VCORE$ not satisfying the above condition.
3. A-cast $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$. Conclude that each $P_j \in VCORE$ is *AWSS-committed* to $(f_j^1(x), \ldots, f_j^\ell(x))$.

VCORE Verification & Agreement on VCORE: Code for P_i — Every party in $\mathcal{P}$, including D, executes this code.

1. Wait to receive $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ from D's A-cast, such that each of the sets are of size at least $2t+1$ and $|VCORE \cap WCORE^{P_j}| \geq 2t+1$.
2. Wait to terminate AWSS-MS-Share$_j$ corresponding to every P_j in $VCORE$.
3. For every $P_j \in VCORE$, wait to receive $\texttt{OK}(P_m, P_k)$ for every $P_m \in OKP_k^{P_j}$ and $P_k \in WCORE^{P_j}$. Then accept $VCORE$, $WCORE^{P_j}$ for $P_j \in VCORE$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$ and terminate WAVSS-MS-Share.

8.4.4 Deciding The Choice of AWSS Protocol

Similar to what was carried out in subsection 8.4.2, here also we analyze the communication complexity of WAVSS-MS while it uses AWSS-MS-I and AWSS-MS-II as black box separately. In our ABA protocol we require a weak statistical AVSS with public reconstruction. This information will be taken into account in our decision for the choice of black box. That is, we will choose the black box as the one which leads to better communication complexity when we consider WAVSS-

Figure 8.8: Reconstruction Phase of our Weak Statistical AVSS Scheme for Sharing Secret S Containing ℓ Elements.

WAVSS-MS-Rec-Private$(D, \mathcal{P}, S = (s^1, \ldots, s^\ell), P_\alpha, \epsilon)$

P_α-weak-private-reconstruction of $(f_j^1(x), \ldots, f_j^\ell(x))$ for every $P_j \in VCORE$: (Code for P_i)

1. Participate in AWSS-MS-Rec-Private$(P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), P_\alpha, \epsilon')$ for every $P_j \in VCORE$ with $WCORE^{P_j}$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$, where $\epsilon' = \frac{\epsilon}{n}$. We denote AWSS-MS-Rec-Private$(P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), P_\alpha, \epsilon')$ by AWSS-MS-Rec-Private$_j$.

Local Computation: Code for P_α

1. For every $P_j \in VCORE$, obtain either $\overline{f_j^1}(x), \ldots, \overline{f_j^\ell}(x)$ or $NULL$ from P_α-weak-private-reconstruction. Add $P_j \in VCORE$ to REC if non-NULL is obtained.
2. Wait until $|REC| = t + 1$. For $l = 1, \ldots, \ell$, do the following:
   (a) For every pair $(P_\gamma, P_\delta) \in REC$ check $\overline{f_\gamma^l(\delta)} \stackrel{?}{=} \overline{f_\delta^l(\gamma)}$. If the test passes for every pair of parties then recover $\overline{F^l}(x, y)$ using $\overline{f_j^l}(x)$'s corresponding to each $P_j \in REC$ and reconstruct $\overline{s^l} = \overline{F^l}(0, 0)$. Else reconstruct $\overline{s^l} = NULL$.
3. For $l = 1, \ldots, \ell$, if any $\overline{s^l} = NULL$ then output $\overline{S} = NULL$ and terminate WAVSS-MS-Rec-Private. Else output $\overline{S} = (\overline{s^1}, \ldots, \overline{s^\ell})$ and terminate WAVSS-MS-Rec-Private.

WAVSS-Rec-Public$(D, \mathcal{P}, s, \epsilon)$

Public reconstruction of $(f_j^1(x), \ldots, f_j^\ell(x))$ for every $P_j \in VCORE$: (Code for P_i)

1. Participate in AWSS-MS-Rec-Public$(P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), \epsilon')$ for every $P_j \in VCORE$ with $WCORE^{P_j}$ and $OKP_k^{P_j}$ for every $P_k \in WCORE^{P_j}$, where $\epsilon' = \frac{\epsilon}{n}$. We denote AWSS-MS-Rec-Public$(P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), \epsilon')$ by AWSS-MS-Rec-Public$_j$.

Local Computation: Code for P_i

For every $P_j \in VCORE$, obtain either $(\overline{f_j^1}(x), \ldots, \overline{f_j^\ell}(x))$ or $NULL$ from public reconstruction. Remaining code is same as the code for P_α presented above for WAVSS-MS-Rec-Private

MS with public reconstruction. In the sequel, we do the analysis and conclude that using AWSS-MS-II as a black box for WAVSS-MS with *public reconstruction* will give us better communication complexity than using AWSS-MS-I as a black box.

Communication complexity of WAVSS-MS using AWSS-MS-I as a Black box: Protocol WAVSS-MS-Share incurs a private communication of $\mathcal{O}((\ell n^3 + n^4 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^3 \log n)$ bits (as WAVSS-MS-Share invokes at most n instances of AWSS-MS-Share). Protocol WAVSS-Rec-Private incurs private communication of $\mathcal{O}((\ell n^3 + n^4 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits (as WAVSS-Rec-Private invokes at most n instances of AWSS-Rec-Private). Protocol WAVSS-Rec-Public incurs A-cast communication of $\mathcal{O}((\ell n^3 + n^4 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits (as WAVSS-Rec-Public invokes at most n instances of AWSS-Rec-Public).

Communication complexity of WAVSS-MS using AWSS-MS-II as a Black box: Protocol WAVSS-MS-Share incurs a private communication of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits. Protocol WAVSS-MS-Rec-Private incurs private communication of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits. Protocol WAVSS-MS-Rec-Public incurs A-cast communication of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits.

So if we consider WAVSS-MS with *public reconstruction* i.e (WAVSS-MS-Share, WAVSS-MS-Rec-Public) then the total communication is better if AWSS-MS-II is used as black box (which is private communication and A-cast communication of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits). So we will consider AWSS-MS-II as a black box for WAVSS-MS and state the communication complexity of WAVSS-MS in the following theorem. Before that we fix the field $\mathbb{F}$ over which WAVSS-MS should work to bound the error probability by ϵ.

To bound the error probability by ϵ, the computation in WAVSS-MS is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^4 2^{-\kappa}$. This is derived from the fact that in WAVSS-MS, AWSS-MS-II will be invoked with $\frac{\epsilon}{n}$ error probability and as mentioned in subsection 8.3.4, $\epsilon \geq n^3 2^{-\kappa}$ should hold to bound error probability of AWSS-MS-II by ϵ. So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

Theorem 8.29 (Communication Complexity of WAVSS-MS) *Using AWSS-MS-II as building block, the communication complexity of WAVSS-MS becomes as follows:*

- *Protocol WAVSS-MS-Share incurs a private communication of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits.*
- *Protocol WAVSS-MS-Rec-Public incurs A-cast of $\mathcal{O}((\ell n^3 + n^4) \log \frac{1}{\epsilon})$ bits.*

PROOF: The proof follows from Lemma 8.18 and the fact that in WAVSS-MS-Share, there can be $\Theta(n)$ instances of AWSS-MS-Share, each executed with an error probability of $\epsilon' = \frac{\epsilon}{n}$. Moreover, in WAVSS-MS-Rec-Public there can be $\Theta(n)$ instances of AWSS-MS-Rec-Public. □

From Theorem 8.29 and 8.25, we can conclude that dealing with multiple secrets concurrently in a protocol provides with better communication complexity that multiple executions of protocol dealing with single secret.

The protocol WAVSS-MS will be used in our ABA protocol presented in Chapter 9. In the next section, we present another AVSS protocol which is a strong statistical AVSS and hence we will use this for AMPC protocol in Chapter 10.

8.5 Our Strong Statistical AVSS protocol

For the sake of simplicity, we first present our strong AVSS protocol sharing a single secret and then extend the protocol for multiple (i.e ℓ) secrets. We will later show that dealing with multiple secrets concurrently in our protocol provides with better communication complexity that multiple executions of protocol dealing with single secret. We may use any one of the AWSS presented in this chapter as a building block for our protocol. But we will use AWSS-I and corresponding multiple secret AWSS protocol AWSS-MS-I for our single and multiple secret version of AVSS respectively. We dedicate a subsection at the end of this section to state the reason for our choice. In the sequel, our AVSS protocols are described without hinting on which AWSS is used, as the AWSS can be replaced by either one of the two AWSS protocols described in Section 8.3. Lastly, our strong statistical AVSS protocol is much more involved than our weak statistical AVSS presented in the previous section.

8.5.1 Our Strong Statistical AVSS Scheme for Sharing a Single Secret

We now present an AVSS scheme called SAVSS consisting of three sub-protocols (SAVSS-Share, SAVSS-Rec-Private, SAVSS-Rec-Public). SAVSS-Share allows D to share a *single secret* from $\mathbb{F}$. Protocol SAVSS-Rec-Private allows a specific party, say P_α, to *privately* reconstruct D's committed secret. We call the private reconstruction as P_α-*private-reconstruction*. While P_α-*private-reconstruction* can always ensure that P_α reconstructs D's committed secret with high probability, P_α-*weak-private-reconstruction* (introduced in Section 8.3) could only ensure that P_α reconstructs either D's committed secret or $NULL$. Protocol SAVSS-Rec-Public enables all the parties in $\mathcal{P}$ to reconstruct D's committed secret.

Structurally, we divide SAVSS-Share into a sequence of following three phases. *Each of the phases will be eventually completed by every honest party when D is honest. Moreover, if some honest party completes all the three phases then eventually every other honest party will also complete all the three phases.*

1. **Commitment by D:** Here D on having a secret s, commits the secret by AWSS-committing n shares of s using n different instances of AWSS-Share protocol.

2. **Verification of D's commitment:** Here the parties verify whether indeed D has committed a secret from $\mathbb{F}$.

3. **Re-commitment by Individual Parties:** If the parties are convinced in previous phase, then every party P_i re-commits his share of D's committed secret using an instance of AWSS-Share protocol.

While first two phases of SAVSS-Share are enough to ensure that D has committed a secret from $\mathbb{F}$, the sole purpose of third phase is to enable robust reconstruction of D's committed secret in SAVSS-Rec-Private and SAVSS-Rec-Public. That is, if protocol SAVSS-Share stops after the second phase, then we may only ensure that either D's committed secret or $NULL$ will be reconstructed in SAVSS-Rec-Private and SAVSS-Rec-Public. This would violate the claim that SAVSS is an AVSS scheme.

8.5.1.1 Commitment by D Phase

In this phase, D on having a secret s, selects a random bivariate polynomial $F(x,y)$ of degree-(t,t) (i.e degree-t in both x and y) such that $F(0,0) = s$. Now to party P_i, D passes $f_i(x) = F(x,i)$ and $g_i(y) = F(i,y)$. We refer $f_i(x)$ polynomials as *row polynomials* and $g_i(y)$ polynomials as *column polynomials*. Now D commits $f_1(x), \ldots, f_n(x)$ using n distinct invocations of AWSS-Share protocol (see Notation 8.11 in Section 8.3.1 for the interpretation of committing polynomial using AWSS-Share). During the course of executing these n instances of AWSS-Share, a party P_i receives i^{th} point on the polynomials $f_1(x), \ldots, f_n(x)$, namely $f_1(i), \ldots, f_n(i)$ which should be n distinct points on $g_i(y)$. So P_i checks whether $g_i(j) = f_j(i)$ for all $j = 1, \ldots, n$ and informs this by A-casting a signal. While executing the n instances of AWSS-Share, D employ a trick to guarantee that all the n instances of AWSS-Share terminate with a common $WCORE$. Once $WCORE$ is agreed among all the honest parties in $\mathcal{P}$, **Commitment by D Phase** ends. The code for this phase is presented in Fig. 8.9.

We now prove the properties of **Commitment by D Phase**.

Lemma 8.30 *In Code Commitment:*

1. *If D is honest then eventually he will generate a common $WCORE$ of size $2t+1$ for all the n instances of AWSS-Share initiated by him. Moreover, each honest party will eventually agree on the common $WCORE$.*

2. *If D is corrupted and some honest party has accepted the $WCORE$ and OKP_js received from the A-cast of D, then every other honest party will also eventually accept the same.*

PROOF: In Code Commitment, D keeps on adding new parties in each $WCORE^i$ and OKP_j^i even after their cardinality reaches $2t+1$. So if D is honest, then eventually he will include all the $2t+1$ honest parties in each $WCORE^i$ and OKP_j^i for every honest P_j. Moreover, each honest P_i will eventually A-cast `Matched-Column` signal, as $f_j(i) = g_i(j)$ will hold for all $j = 1, \ldots, n$ when D is *honest*. Therefore, if D is honest then eventually he will find a common set of at least $2t+1$ parties in the $WCORE^i$ of all the n instances of AWSS-Share, who have A-cast `Matched-Column` signal. This common set of at least $2t+1$ parties will form the common $WCORE$ which D will A-cast. Similarly, corresponding to each $P_j \in WCORE$, the honest D will eventually find a common set of at least $2t+1$ parties in $OKP_j^1, \ldots, OKP_j^n$. This common set of at least $2t+1$ parties will constitute OKP_j which D will A-cast. Now from the property of A-cast, each honest party will eventually receive $WCORE$ of size at least $2t+1$ and OKP_j of size at least $2t+1$ for each $P_j \in WCORE$ from the A-cast of D. Now it is easy to see that each honest party will accept this common $WCORE$ and OKP_j for each $P_j \in WCORE$, after executing the steps in [**WCORE verification and agreement**]. This proves the first part.

If D is corrupted and some honest party, say P_i has accepted the $WCORE$ and OKP_j's received from the A-cast of D then it implies the following: P_i has received $OK(P_k, P_j)$ from the A-cast of P_k for every $P_k \in OKP_j$ and every $P_j \in WCORE$ for all the n executions of AWSS-Share. Moreover, P_i has also received `Matched-Column` from A-cast of every $P_j \in WCORE$. Now from the property of A-cast, every other honest party P_j will also eventually receive the same `OK`s and `Matched-Column` and hence will accept the $WCORE$ and OKP_j for each $P_j \in WCORE$.

Figure 8.9: Code for Commitment by D Phase

Code Commitment($D, \mathcal{P}, s, \epsilon$)

i. Distribution by D: – Only D executes this code

1. Select a random degree-(t, t) bivariate polynomial $F(x, y)$ such that $F(0, 0) = s$.
2. For $i = 1, \ldots, n$, send *row polynomial* $f_i(x) = F(x, i)$ and *column polynomial* $g_i(y) = F(i, y)$ to P_i.
3. For $i = 1, \ldots, n$, initiate AWSS-Share$(D, \mathcal{P}, f_i(x), \epsilon')$ (see Notation 8.11 for syntax) for sharing $f_i(x)$, where $\epsilon' = \frac{\epsilon}{n}$.

ii. Code for P_i: – Every party in $\mathcal{P}$, including D, executes this code

1. Wait to receive degree-t row polynomial $f_i(x)$ and degree-t column polynomial $g_i(y)$ from D.
2. Participate in AWSS-Share$(D, \mathcal{P}, f_j(x), \epsilon')$ by executing steps in [**Verification: Code for P_i**] (of AWSS-Share) for all $j = 1, \ldots, n$.
3. After the completion of step 1 of [**Verification: Code for P_i**] for all the n invocations of AWSS-Share, check whether $g_i(j) = f_j(i)$ holds for all $j = 1, \ldots, n$. Here $f_j(i)$ is obtained by P_i from D during the execution of first step of [**Verification: Code for P_i**] of AWSS-Share$(D, \mathcal{P}, f_j(x), \epsilon')$. If yes then A-cast `Matched-Column` and execute the rest of the steps of AWSS-Share$(D, \mathcal{P}, f_j(x), \epsilon')$, for all $j = 1, \ldots, n$.

iii. WCORE Construction: Code for D – Only D executes this code.

1. Construct $WCORE$ and corresponding OKP_j's for each AWSS-Share$(D, \mathcal{P}, f_i(x), \epsilon')$ following the steps in [**WCORE Construction**] (of AWSS-Share). Denote them by $WCORE^i$ and OKP_j^i's.
2. Keep updating $WCORE^i$'s and corresponding OKP_j^i's. That is, even after $WCORE^i$ reaches the size of $2t + 1$, D keeps on adding new parties in $WCORE^i$ and corresponding OK sets after receiving new A-cast of the form `OK(*, *)` in AWSS-Share$(D, \mathcal{P}, f_i(x), \epsilon')$.
3. Wait to obtain $WCORE = \cap_{i=1}^{n} WCORE^i$ of size at least $2t + 1$ and for every $P_j \in WCORE, OKP_j = \cap_{i=1}^{n} OKP_j^i$ of size at least $2t+1$, such that `Matched-Column` is received from A-cast of every $P_j \in WCORE$.
4. A-cast $WCORE$ and OKP_j for every $P_j \in WCORE$.

iv. WCORE verification & Agreement: Code for P_i – Every party including D will execute this code.

1. Wait to receive $WCORE$ and OKP_j for every $P_j \in WCORE$ from A-cast of D, such that $|WCORE| \geq 2t + 1$ and each $|OKP_j| \geq 2t + 1$.
2. Wait to receive $OK(P_k, P_j)$ from the A-cast of P_k for every $P_k \in OKP_j$ and every $P_j \in WCORE$ for all the n executions of AWSS-Share.
3. Wait to receive `Matched-Column` from A-cast of every $P_j \in WCORE$.
4. After receiving all desired `OK`s and `Matched-Column` signals, accept $WCORE$ and OKP_j for every $P_j \in WCORE$ received from A-cast of D and proceed to the next phase (**Verification of D's Commitment Phase**).

8.5.1.2 Verification of D's Commitment Phase

After agreeing on $WCORE$ and corresponding OKP_j's, in this phase, the parties verify whether indeed D has committed a secret from $\mathbb{F}$. For this, the parties try to check whether there is a set $\mathcal{R}$ of size at least $2t + 1$ and another set $\mathcal{C}$ of size at least $2t + 1$ (possibly different from $\mathcal{R}$), such that for every $P_i \in \mathcal{R}$ and every $P_j \in \mathcal{C}$, $f_i(j) = g_j(i)$ holds. If they can ensure that such sets exist then it implies that the row and column polynomials of the *honest* parties in $\mathcal{R}$ and $\mathcal{C}$ define a unique bivariate polynomial of degree-(t,t) and the constant term of the polynomial is D's committed secret. Checking for the existence of such sets is quiet easy in synchronous settings, where the parties can simply pair-wise exchange common values on their row and column polynomial, as done in several synchronous VSS protocols [20, 91, 73, 109, 125]. However, doing the same is not so straightforward in asynchronous settings, especially when we have only $3t+1$ parties.

To check the existence of the sets described above, the parties proceed as follows: recall that in the **Commitment by D phase**, D is committed to $f_1(x), \ldots, f_n(x)$ using n distinct instances of AWSS-Share. Now the parties execute AWSS-Rec-Private$(D, \mathcal{P}, f_j(x),$
$P_j, \epsilon')$ for enabling P_j-weak-private-reconstruction of $f_j(x)$. If P_j reconstructs $\overline{f_j}(x)$ from the execution of AWSS-Rec-Private and $\overline{f_j}(x)$ is same as $f_j(x)$ received from D in the previous phase, then P_j informs this to everyone by A-casting `Matched-Row` signal. This is a public notification by P_j that the polynomial committed by D in AWSS-Share$(D, \mathcal{P}, f_j(x), P_j, \epsilon')$ is same as the one which P_j has privately received from D. Now if at least $2t+1$ parties, say $\mathcal{R}$, A-cast `Matched-Row`, then it implies that D is committed to a unique degree-(t,t) bivariate polynomial, say $\overline{F}(x,y)$ (hence a unique secret $\overline{s} = \overline{F}(0,0)$) such that for every *honest* $P_i \in \mathcal{R}$, the row polynomial $f_i(x)$ held by P_i satisfies $\overline{F}(x,i) = f_i(x)$ and for every *honest* $P_j \in WCORE$, the column polynomial $g_j(y)$ held by P_j satisfies $\overline{F}(j,y) = g_j(y)$ (For proof see Lemma 8.31). The code for implementing this phase is very easy and is given in Fig. 8.10.

Lemma 8.31 *In Code Verification, if an honest party receives* `Matched-Row` *from the A-cast of the parties in $\mathcal{R}$, then in code Commitment, D is committed to a unique degree-(t,t) bivariate polynomial $\overline{F}(x,y)$ (and hence to a secret $\overline{s} = \overline{F}(0,0)$) such that the row polynomial $f_i(x)$ held by every honest $P_i \in \mathcal{R}$ satisfies $\overline{F}(x,i) = f_i(x)$ and the column polynomial $g_j(y)$ held by every honest $P_j \in WCORE$ satisfies $\overline{F}(j,y) = g_j(y)$. Moreover if D is honest then $\overline{F}(x,y) = F(x,y)$ and hence $\overline{s} = s$.*

PROOF: Let l and m be the number of honest parties in $\mathcal{R}$ and $WCORE$ respectively. As $|WCORE| \geq 2t+1$ and $|\mathcal{R}| \geq 2t+1$, both $l \geq t+1$ and $m \geq t+1$. For convenience, we assume $P_1, \ldots, P_l$ and respectively $P_1, \ldots, P_m$ are the set of honest parties in $\mathcal{R}$ and $WCORE$. Now for every (P_i, P_j) with $P_i \in \{P_1, \ldots, P_l\}$ and $P_j \in \{P_1, \ldots, P_m\}$, $f_i(j) = g_j(i)$ holds. This is due to the fact that P_i has checked that D is indeed committed to $f_i(x)$ (by checking $f_i(x) = \overline{f_i}(x)$, where $\overline{f_i}(x)$ is obtained from P_i-weak-private-reconstruction and $f_i(x)$ is obtained from D in Commitment). The above implies that honest $P_j \in WCORE$ has received $f_i(j)$ from D and checked $g_j(i) = f_i(j)$ during the execution of Commitment. We now claim that if $f_i(j) = g_j(i)$ holds for every (P_i, P_j) with $P_i \in \{P_1, \ldots, P_l\}$ and $P_j \in \{P_1, \ldots, P_m\}$ then there exists a unique bivariate polynomial $\overline{F}(x,y)$ of degree-(t,t), such that for $i = 1, \ldots, l$, we have $\overline{F}(x,i) = f_i(x)$ and for $j = 1, \ldots, m$, we have $\overline{F}(j,y) = g_j(y)$. The proof now completely follows from the proof of Lemma 4.26 of [35].

Specifically, let $V^{(k)}$ denote $k \times k$ Vandermonde matrix, where i^{th} column is $[i^0, \ldots, i^{k-1}]^T$, for $i = 1, \ldots, k$. Now consider the degree-t row polynomials $f_1(x), \ldots, f_{t+1}(x)$ and let E be the $(t+1) \times (t+1)$ matrix, where E_{ij} is the coefficient of x^j in $f_i(x)$, for $i = 1, \ldots, t+1$ and $j = 0, \ldots, t$. Thus for $i =$

Figure 8.10: Code for Verification of D's Commitment Phase

Code Verification$(D, \mathcal{P}, s, \epsilon)$

i. P_j-Weak-Private-Reconstruction of $f_j(x)$ for $j = 1, \ldots, n$: CODE FOR P_i – Every party in $\mathcal{P}$ executes this code.

1. After agreeing on $WCORE$ and corresponding OKP_j's, participate in AWSS-Rec-Private$(D, \mathcal{P}, f_j(x), P_j, \epsilon')$, for $j = 1, \ldots, n$, to enable P_j-weak-private-reconstruction of $f_j(x)$. Notice that the same $WCORE$ and OKP_j for $P_j \in WCORE$ are used in each AWSS-Rec-Private$(D, \mathcal{P}, f_j(x), P_j, \epsilon')$, for $j = 1, \ldots, n$
2. At the completion of AWSS-Rec-Private$(D, \mathcal{P}, f_i(x), P_i, \epsilon')$, obtain either degree-$t$ polynomial $\overline{f_i}(x)$ or $NULL$.
3. If $f_i(x) = \overline{f_i}(x)$, then A-cast `Matched-Row`.

ii. Code for D: — Only executes this code.

1. Wait to receive `Matched-Row` from A-cast of at least $2t + 1$ parties, say $\mathcal{R}$.
2. A-cast the set $\mathcal{R}$.

iii. Verification of D's Commitment: Code for P_i – Every party in $\mathcal{P}$ executes this code

1. Wait to receive $\mathcal{R}$ of size at least $2t + 1$ from A-cast of D.
2. Wait to receive `Matched-Row` from A-cast of every party in $\mathcal{R}$ and then proceed to third phase.

$1, \ldots, t+1$ and $j = 1, \ldots, t+1$, the $(i, j)^{th}$ entry in $E \cdot V^{(t+1)}$ is $f_i(j)$.

Let $H = ((V^{(t+1)})^T)^{-1} \cdot E$ be a $(t+1) \times (t+1)$ matrix. Let for $i = 0, \ldots, t$, the $(i+1)^{th}$ column of H be $[r_{i0}, r_{i1}, \ldots, r_{it}]^T$. Now we define a degree-(t, t) bivariate polynomial $\overline{F}(x, y) = \sum_{i=0}^{i=t} \sum_{j=0}^{j=t} r_{ij} x^i y^j$. Then from properties of bivariate polynomial, for $i = 1, \ldots, t+1$ and $j = 1, \ldots, t+1$, we have

$$\overline{F}(j, i) = (V^{(t+1)})^T \cdot H \cdot V^{(t+1)} = E \cdot V^{(t+1)} = f_i(j) = g_j(i)$$

This implies that for $i = 1, \ldots, t+1$, the polynomials $\overline{F}(x, i)$ and $f_i(x)$ have same value at $t+1$ values of x. But since degree of $\overline{F}(x, i)$ and $f_i(x)$ is t, this implies that $\overline{F}(x, i) = f_i(x)$. Similarly, for $j = 1, \ldots, t+1$, we have $\overline{F}(j, y) = g_j(y)$.

Next, we will show that for any $t+1 < i \leq l$, the polynomial $f_i(x)$ also lies on $\overline{F}(x, y)$. In other words, $\overline{F}(x, i) = f_i(x)$, for $t+1 < i \leq l$. This is easy to show because according to theorem statement, $f_i(j) = g_j(i)$, for $j = 1, \ldots, t+1$ and $g_1(i), \ldots, g_{t+1}(i)$ lie on $\overline{F}(x, i)$ and uniquely defines $\overline{F}(x, i)$. Since both $f_i(x)$ and $\overline{F}(x, i)$ are of degree t, this implies that $\overline{F}(x, i) = f_i(x)$, for $t+1 < i \leq l$. Similarly, we can show that $\overline{F}(j, y) = g_j(y)$, for $t+1 < j \leq m$. The second part of the lemma is trivially true. □

Lemma 8.32 *In Code Verification, if D is honest then all honest parties will eventually proceed to third phase, except with probability ϵ. Moreover, if D is* *corrupted and some honest party proceeds to the third phase, then all other honest party will also eventually proceed to the third phase.*

PROOF: If D is honest then every honest party P_i will eventually A-cast `Matched-Row` signal, except with probability ϵ'. The reason is that every honest P_i will privately reconstruct back $f_i(x)$ in AWSS-Rec-Private$(D, \mathcal{P}, f_i(x), P_i, \epsilon')$, except with probability ϵ'. As there are at least $2t+1$ honest parties, except with probability at most $n\epsilon' = \epsilon$, all honest parties will eventually A-cast `Matched-Row` signal. Therefore, eventually there will be a set of $2t+1$ parties, say $\mathcal{R}$, who will A-cast `Matched-Row` signal. D will A-cast $\mathcal{R}$ and eventually every honest party will receive $\mathcal{R}$ from A-cast of D and will eventually receive $2t+1$ `Matched-Row` signal from the parties of $\mathcal{R}$ and will proceed to the third phase, except with probability ϵ.

If D is corrupted and some honest party P_i proceeds to the third phase, then it implies that P_i has received $\mathcal{R}$ from A-cast of D and then `Matched-Row` signal from every party in $\mathcal{R}$. So eventually all other honest parties will also receive $\mathcal{R}$ and corresponding `Matched-Row` signals and will proceed to the third phase. □

From Lemma 8.31, if the honest parties agree on $\mathcal{R}$, then they are sure that D is committed to a unique bivariate polynomial and thus a unique secret. Now the question is: *If the honest parties stop protocol SAVSS-Share after agreeing on $\mathcal{R}$, then is there any possible way of robustly reconstructing D's secret in reconstruction phase*? Here we stop a moment and try to find the possibilities for the above question. Our effort in this direction would also motivate the need of the third phase of SAVSS-Share which is actually required to enable robust reconstruction of D's committed secret in the reconstruction phase i.e in SAVSS-Rec-Private (and SAVSS-Rec-Public).

One possible way to reconstruct D's committed secret s is to execute AWSS-Rec-Private$(D, \mathcal{P}, f_j(x), *, \epsilon')$ corresponding to every $P_j \in \mathcal{R}$, which may disclose $f_j(x)$ polynomials and using those polynomial the bivariate polynomial and thus the secret s may be reconstructed. But this does not work, because when D is *corrupted*, *all* instances of AWSS-Rec-Private may output $NULL$. So it seems that most likely there is no way to robustly reconstruct D's committed value s in protocol SAVSS-Rec-Private (and SAVSS-Rec-Public), if SAVSS-Share stops after second phase. Hence, we require the third phase which is described in the sequel. Prior to the description of the third phase in the next section, we present the following remark.

Remark 8.33 *In the code* **Commitment**, *D executed n instances of AWSS-Share for individually committing each $f_i(x)$. Later this allowed $f_i(x)$ to be privately reconstructed only by P_i during the code for* **Verification**. *If D executes a single instance of AWSS-MS-Share for concurrently committing to $f_1(x), \ldots, f_n(x)$, instead of n instances of AWSS-Share, then later in the code* **Verification**, *we could not enable P_1 to privately reconstruct $f_1(x)$, P_2 to privately reconstruct $f_2(x)$ and so on. This is because AWSS-MS-Rec-Private is designed in such a way that it will allow P_α to privately reconstruct back all the n polynomials. This would clearly breach the secrecy property of AVSS as every party will now come to know all the n row polynomials.*

8.5.1.3 Re-commitment by Individual Parties

The outline for this phase is as follows: If P_i A-casts `Matched-Row` in code Verification, then P_i acts as a dealer to re-commit his row polynomial $f_i(x)$ by initiating an instance of AWSS-Share. *It is also enforced that if P_i attempts to re-commit $f'_i(x) \neq f_i(x)$, then his re-commitment will not be terminated.* Moreover, when D is honest then an honest P_i will always be able to successfully re-commit $f_i(x)$. Now SAVSS-Share terminates only when all the honest parties in $\mathcal{P}$ agree upon a set of at least $2t+1$ parties, say $VCORE$, who have successfully re-committed their polynomials. Now clearly, if SAVSS-Share terminates, then the robust reconstruction of $D's$ committed secret s is guaranteed with very high probability later in reconstruction phase. This is because, the AWSS-Rec-Private instance of an *honest* $P_i \in VCORE$ will always reconstruct back $f_i(x)$. On the other hand, AWSS-Rec-Private instance of a *corrupted* $P_i \in VCORE$ will output either $f_i(x)$ or $NULL$ with probability at least $(1-\epsilon')$. This guarantees the reconstruction of at least $t+1$ $f_i(x)$ polynomials which are enough to reconstruct D's committed bivariate polynomial and hence s. The protocol for this phase is given in Fig. 8.11.

Lemma 8.34 *If D is honest then D will eventually generate $VCORE$ of size $2t+1$, except with probability ϵ. Moreover each honest party will agree on this $VCORE$. If D is corrupted and some honest party has accepted $VCORE$ received from D, then every other honest party will also eventually do the same.*

Proof: From the proof of Lemma 8.32, if D is honest, then all honest parties (at least $2t+1$) will eventually A-cast `Matched-Row` in code Verification, except with probability ϵ. So except with probability ϵ, all these honest P_i's will eventually complete AWSS-Share$_i$ as a dealer and thus will re-commit $f_i(x)$ successfully. Therefore, D will eventually find a set of $2t+1$ P_i's for which the conditions stated in step 1 of [**VCORE Construction**] will be eventually satisfied. Hence D will add all these $2t+1$ P_i's in $VCORE$ and A-cast the same. Now it is easy to see that every honest party will agree on this $VCORE$ after performing the steps in [**VCORE Verification & Agreement on VCORE**].

If D is corrupted and some honest party P_i has accepted $VCORE$ received from D, then it implies that P_i has checked the validity of received $VCORE$ by performing the steps in [**VCORE Verification & Agreement on VCORE**]. Now it is easy to see that all other honest parties will also do the same and will accept $VCORE$ eventually. □

Lemma 8.35 *If $VCORE$ is generated, then there exists a unique degree-(t,t) bivariate polynomial $\overline{F}(x,y)$ such that every $P_i \in VCORE$ is re-committed to $f_i(x) = \overline{F}(x,i)$. Moreover, if D is honest then $\overline{F}(x,y) = F(x,y)$.*

Proof: By Lemma 8.31, there is a unique degree-(t,t) bivariate polynomial $\overline{F}(x,y)$ such that the row polynomial of every *honest* P_i who has A-casted `Matched-Row`, satisfies $f_i(x) = \overline{F}(x,i)$. Since an honest party P_i who has re-committed his row polynomial $f_i(x)$ in Re-commitment, has also A-casted `Matched-Row` in Verification, $f_i(x) = \overline{F}(x,i)$ satisfies for every *honest* P_i in $VCORE$. Now we show that even a *corrupted* $P_i \in VCORE$ has re-committed $f_i(x)$ satisfying $f_i(x) = \overline{F}(x,i)$.

We prove this by showing that every *honest* $P_j \in WCORE^{P_i}$ has received $f_i(j)$ from P_i during AWSS-Share$_i$ (and hence honest P_j is *IC-committed* to $f_i(j)$). An honest P_j belongs to $WCORE^{P_i}$ implies that P_j belongs to $ProbCORE$ of at

Figure 8.11: Code for "Re-commitment by Individual Parties" Phase

Code Re-commitment($D, \mathcal{P}, s, \epsilon$)

i. Code for P_i: — Every party executes this code.

1. If you have A-casted `Matched-Row` in Verification then as a dealer, initiate AWSS-Share($P_i, \mathcal{P}, f_i(x), \epsilon'$) to re-commit $f_i(x)$ with $\epsilon' = \frac{\epsilon}{n}$. We denote this instance by AWSS-Share$_i$.
2. If P_j has A-casted `Matched-Row` in Verification, then participate in AWSS-Share$_j$ by executing steps in [**Verification: Code for** P_i] (of AWSS-Share) in the following way:

   After the completion of step 1 of [**Verification: Code for** P_i], check whether $g_i(j) = f_j(i)$ holds, where $f_j(i)$ is obtained from the execution of AWSS-Share$_j$ and $g_i(y)$ was obtained from D during **commitment by** D **phase**. If yes then participate in the remaining steps of [**Verification: Code for** P_i] corresponding to AWSS-Share$_j$.
3. $WCORE^{P_i}$ **Construction for** AWSS-Share$_i$: If P_i initiated AWSS-Share$_i$ to re-commit $f_i(x)$, then P_i as a dealer, constructs $WCORE$ and corresponding OKP_js for AWSS-Share$_i$ in a slightly different way than what is described in AWSS-Share (*these steps also ensure that a corrupted* P_i *will not be able to re-commit* $\overline{f_i}(x) \neq f_i(x)$).
   (a) Construct a set $ProbCORE^{P_i}$ ($= \emptyset$ initially). Include P_j in $ProbCORE^{P_i}$ and A-cast $(P_j, ProbCORE^{P_i})$ if at least $2t+1$ A-casts of the form $\mathtt{OK}(., P_j)$ are heard in the instance AWSS-Share$_i$.
   (b) Construct $WCORE^{P_i}$. Add P_j in $WCORE^{P_i}$ if both the following holds:
      (A) $P_j \in ProbCORE^{P_i}$ and
      (B) If $(P_j, ProbCORE^{P_k})$ is received from the A-cast of at least $2t+1$ P_k's who have A-casted `Matched-Row`.
   (c) A-cast $WCORE^{P_i}$ and OKP_j for every $P_j \in WCORE^{P_i}$ when $|WCORE^{P_i}| = 2t+1$.

ii. VCORE Construction: Code for D — Only D executes this code

1. If $WCORE^{P_i}$ and OKP_j for every $P_j \in WCORE^{P_i}$ are received from the A-cast of P_i, then add P_i to $VCORE$ after performing the following:
   (a) Wait to receive $(P_j, ProbCORE^{P_i})$ for every $P_j \in WCORE^{P_i}$ from the A-cast of P_i.
   (b) Wait to receive $(P_j, ProbCORE^{P_k})$ for every $P_j \in WCORE^{P_i}$ from A-cast of at least $2t+1$ P_k's who have A-casted `Matched-Row`.
   (c) Wait to receive $OK(P_j, P_k)$ for every $P_k \in OKP_j$ in execution AWSS-Share$_i$.
2. A-cast $VCORE$ when $|VCORE| = 2t+1$.

iii. VCORE Verification & Agreement on VCORE: Code for P_i — Every party executes this code

1. Wait to receive $VCORE$ from the A-cast of D.
2. For every $P_i \in VCORE$, wait to receive $WCORE^{P_i}$ and OKP_j for every $P_j \in WCORE^{P_i}$ from the A-cast of P_i.
3. Once received, check the validity of received $WCORE^{P_i}$'s and OKP_j's for every $P_j \in WCORE^{P_i}$ by following the same steps as in ii-1(a), ii-1(b) and ii-1(c).
4. After checking the validity, accept (i) $VCORE$; (ii) $WCORE^{P_i}$ and corresponding OKP_j's for every $P_i \in VCORE$ which are received in previous two steps and terminate SAVSS-Share.

least $2t+1$ parties (who have A-casted `Matched-Row`) out of which at least $t+1$ are honest. Let $\mathcal{H}$ be the set of these $(t+1)$ honest parties. So P_j's column polynomial $g_j(y)$ satisfies $g_j(k) = f_k(j)$ for every $P_k \in \mathcal{H}$ (due to step i-(2) in Re-commitment). This implies that $g_j(y) = \overline{F}(j, y)$. Now honest $P_j \in WCORE^{P_i}$ implies that P_j belongs to $ProbCORE$ of P_i as well which means P_j has ensured $g_j(i) = f_i(j)$ (due to step i-(2)) in Re-commitment.

Now the second part of the lemma is trivially true. □

8.5.1.4 Protocol SAVSS

Now the protocols for our AVSS scheme is presented in Fig. 8.12.

Figure 8.12: Our Strong Statistical AVSS for Sharing Secret s with $n = 3t + 1$

Protocol SAVSS($D, \mathcal{P}, s, \epsilon$)

SAVSS-Share($D, \mathcal{P}, s, \epsilon$)

1. Replicate Code Commitment($D, \mathcal{P}, s, \epsilon$).
2. Replicate Code Verification($D, \mathcal{P}, s, \epsilon$).
3. Replicate Code Re-commitment($D, \mathcal{P}, s, \epsilon$).

SAVSS-Rec-Private($D, \mathcal{P}, s, P_\alpha, \epsilon$): P_α-private-reconstruction of s:

P_α-weak-private-reconstruction of $f_j(x)$ for every $P_j \in VCORE$: (Code for P_i)

1. Participate in AWSS-Rec-Private($P_j, \mathcal{P}, f_j(x), P_\alpha, \epsilon'$) for every $P_j \in VCORE$. We denote AWSS-Rec-Private($P_j, \mathcal{P}, f_j(x), P_\alpha, \epsilon'$) by AWSS-Rec-Private$_j$

Local Computation: Code for P_α

1. For every $P_j \in VCORE$, obtain either $\overline{f_j}(x)$ or $NULL$ from P_α-weak-private-reconstruction. Add $P_j \in VCORE$ to REC if $\overline{f_j}(x)$ is obtained.
2. Wait until $|REC| = t + 1$. Construct bivariate polynomial $\overline{F}(x, y)$ such that $\overline{F}(x, j) = \overline{f_j}(x)$ for every $P_j \in REC$. Compute $\overline{s} = \overline{F}(0, 0)$ and terminate SAVSS-Rec-Private.

SAVSS-Rec-Public($D, \mathcal{P}, s, \epsilon$): Public reconstruction of s:

Public reconstruction of $f_j(x)$ for every $P_j \in VCORE$: (Code for P_i)

1. Participate in AWSS-Rec-Public($P_j, \mathcal{P}, f_j(x), \epsilon'$) for every $P_j \in VCORE$. We denote AWSS-Rec-Public($P_j, \mathcal{P}, f_j(x), \epsilon'$) by AWSS-Rec-Public$_j$

Local Computation: Code for P_i

Same code as presented above for P_α in SAVSS-Rec-Private.

In the following, we prove the properties of our AVSS scheme considering SAVSS-Rec-Private as the reconstruction phase protocol. The proofs can be twisted to obtain proofs for our AVSS considering SAVSS-Rec-Public as reconstruction phase protocol.

Lemma 8.36 (AVSS-Termination) *Protocol SAVSS satisfies* **termination** *property of Definition 8.2.*

Proof: **Termination 1** and **Termination 2** property follows from Lemma 8.30, Lemma 8.32 and Lemma 8.34. **Termination 3** property is proved as follows: By **Termination 3** and **Correctness 1** of our AWSS scheme (see Lemma 8.7 and Lemma 8.9), AWSS-Share$_i$ initiated by an *honest* P_i in $VCORE$ during Re-commitment, will reconstruct $f_i(x)$ in its reconstruction phase, except with probability at most ϵ' (In SAVSS, the AWSS schemes were executed with error probability $\epsilon' \approx \frac{\epsilon}{n}$). But AWSS-Share$_i$ initiated by a *corrupted* P_i in $VCORE$, may lead to the reconstruction of $NULL$ in its reconstruction phase. Since $|VCORE| = 2t+1$, for at least $t+1$ honest parties in $VCORE$, reconstruction of $f_i(x)$'s will be successful, except with probability $(t+1)\epsilon' \approx \epsilon$. This is enough to reconstruct the secret s. Hence if all honest parties terminate SAVSS-Share and every (honest) party starts SAVSS-Rec-Private, then an honest P_α will eventually terminate SAVSS-Rec-Private with probability at least $(1-\epsilon)$. □

Lemma 8.37 (AVSS-Secrecy) *Protocol SAVSS satisfies secrecy property of Definition 8.2.*

Proof: We have to consider the case when D is honest. Without loss of generality, assume that $P_1, \ldots, P_t$ are the parties under the control of $\mathcal{A}_t$. So throughout SAVSS-Share, $\mathcal{A}_t$ will know $f_1(x), \ldots, f_t(x), g_1(y), \ldots, g_t(y)$ and t points on $f_{t+1}(x), \ldots, f_n(x)$. Moreover, honest parties only exchange common values on their row and column polynomials and by the secrecy property of AWSS-Share, these values will be unknown to $\mathcal{A}_t$. Hence by the property of bivariate polynomial of degree-(t,t) [46], $\mathcal{A}_t$ will lack one more point to uniquely interpolate $F(x,y)$. Hence $s = F(0,0)$ will be information theoretically secure. □

Lemma 8.38 (AVSS-Correctness) *Protocol SAVSS satisfies correctness property of Definition 8.2.*

Proof: By Lemma 8.35, there is a unique degree-(t,t) bivariate polynomial $\overline{F}(x,y)$ such that every $P_i \in VCORE$ has re-committed $f_i(x) = \overline{F}(x,i)$. Moreover, if D is honest then $\overline{F}(x,y) = F(x,y)$. Now by Lemma 8.9, in AWSS-Rec-Private$_i$, except with probability ϵ' the following will happen:

1. For every honest $P_i \in VCORE$, $f_i(x)$ will be reconstructed;

2. For every corrupted $P_i \in VCORE$, $f_i(x)$ or $NULL$ will be reconstructed.

As $|VCORE| = 2t+1$, for at least $t+1$ parties P_i's in $VCORE$, $f_i(x)$ will be reconstructed with probability at least $(1-(t+1)\epsilon') \approx \epsilon$. Using those polynomials $\overline{F}(x,y)$ and $\overline{s} = \overline{F}(0,0)$ will be reconstructed with probability $1-\epsilon$. Moreover, $s = \overline{s} = F(0,0)$ if D is honest. □

Theorem 8.39 *Protocol SAVSS consisting of (SAVSS-Share, SAVSS-Rec-Private, SAVSS-Rec-Public) constitutes a valid strong statistical AVSS scheme.*

Proof: The proof follows from Lemma 8.36, Lemma 8.37 and Lemma 8.38. □

Remark 8.40 (**D's commitment in SAVSS-Share**) *We say that D has committed secret $s \in \mathbb{F}$ in SAVSS-Share if there is a degree-t univariate polynomial, $f(x)$, such that $f(0) = s$ and every honest P_i in $VCORE$ receives $f(i)$ from D and commits to $f(i)$ using AWSS-Share. In protocol SAVSS-Share, $f(x) = f_0(x) = \overline{F}(x, 0)$, where $\overline{F}(x, y)$ is D's committed bivariate polynomial. When D is honest, $\overline{F}(x, y) = F(x, y)$.*

Notation 8.41 (Notation for Using SAVSS-Share and SAVSS-Rec-Private) *Later we will invoke SAVSS-Share as SAVSS-Share($D, \mathcal{P}, f(x), \epsilon$) to mean that D commits $f(x)$ in SAVSS-Share. Essentially here D is asked to choose bivariate polynomial $F(x, y)$ of degree-(t, t) such that $F(x, 0) = f(x)$ holds. Similarly, SAVSS-Rec-Private will be invoked as SAVSS-Rec-Private($D, \mathcal{P}, f(x), P_\alpha, \epsilon$) to enable P_α-private-reconstruction of $f(x)$.*

8.5.2 Deciding The Choice of AWSS Protocol

For our AMPC protocol presented in Chapter 10, we require a strong statistical AVSS with P_α-private-reconstruction. We now need to examine which AWSS will fit better in this case in terms of communication complexity. For this we now analyze the communication complexity of our SAVSS protocol by substituting AWSS-I and AWSS-II.

Communication complexity of SAVSS using AWSS-I as a Black box: Protocol SAVSS-Share incurs a private communication of $\mathcal{O}(n^4(\log\frac{1}{\epsilon})^2)$ bits and A-cast of $\mathcal{O}(n^3 \log n)$ bits (as it requires $\mathcal{O}(n)$ executions of AWSS-Share and AWSS-Rec-Private). Protocol SAVSS-Rec-Private incurs private communication of $\mathcal{O}(n^4(\log\frac{1}{\epsilon})^2)$ bits (as it requires $\mathcal{O}(n)$ executions of AWSS-Rec-Private). Protocol SAVSS-Rec-Public incurs A-cast communication of $\mathcal{O}(n^4(\log\frac{1}{\epsilon})^2)$ bits (as it requires $\mathcal{O}(n)$ executions of AWSS-Rec-Public).

Communication complexity of SAVSS using AWSS-II as a Black box: Protocol SAVSS-Share incurs a private communication of $\mathcal{O}(n^4 \log\frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^4 \log\frac{1}{\epsilon})$ bits. Protocol SAVSS-Rec-Private incurs private communication of $\mathcal{O}(n^4 \log\frac{1}{\epsilon})$ bits. Protocol SAVSS-Rec-Public incurs A-cast communication of $\mathcal{O}(n^4 \log\frac{1}{\epsilon})$ bits.

So if we consider SAVSS with P_α-private-reconstruction i.e (SAVSS-Share,SAVSS-Rec-Private) then the total communication is better when AWSS-I is used as black box (which is private communication of $\mathcal{O}(n^4(\log\frac{1}{\epsilon})^2)$ bits and A-cast of $\mathcal{O}(n^3 \log n)$ bits). This is because for A-casting a single bit requires $\mathcal{O}(n^2)$ bits of private communication [29]. So communication complexity of SAVSS using AWSS-II will be $\mathcal{O}(n^6 \log\frac{1}{\epsilon})$ bits of private communication. So we will consider AWSS-I as a black box for SAVSS and state the communication complexity of SAVSS in the following theorem. Before that we fix the field $\mathbb{F}$ over which SAVSS should work to bound the error probability by ϵ.

To bound the error probability by ϵ, the computation in SAVSS is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^4\kappa 2^{-\kappa}$. This is derived from the fact that in SAVSS, AWSS-I is invoked with $\frac{\epsilon}{n}$ error probability and as mentioned in subsection 8.3.3, $\epsilon \geq n^3\kappa 2^{-\kappa}$ should hold to bound error probability of AWSS-I by ϵ. So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

Theorem 8.42 (Communication Complexity of SAVSS) *Using AWSS-I as building block, the communication complexity of SAVSS becomes as follows:*

- *Protocol SAVSS-Share incurs a private communication of $\mathcal{O}(n^4(\log \frac{1}{\epsilon})^2)$ bits and A-cast of $\mathcal{O}(n^3 \log n)$ bits.*
- *Protocol SAVSS-Rec-Private incurs a private communication of $\mathcal{O}(n^4(\log \frac{1}{\epsilon})^2)$ bits.*

PROOF: The proof follows from Lemma 8.15 and the fact that SAVSS-Share invokes $\Theta(n)$ instances of AWSS-Share and AWSS-Rec-Private, each with an error probability of $\epsilon' = \frac{\epsilon}{n}$. Moreover, WAVSS-Rec-Private invokes $\Theta(n)$ instances of AWSS-Rec-Private. □

8.5.3 Our Strong Statistical AVSS Scheme for Sharing Multiple Secrets

We now present a strong statistical AVSS scheme SAVSS-MS, consisting of protocols (SAVSS-MS-Share, SAVSS-MS-Rec-Private,SAVSS-MS-Rec-Public). Protocol SAVSS-MS-Share allows D to share a secret $S = (s^1, \ldots, s^\ell)$, consisting of $\ell > 1$ elements from $\mathbb{F}$. While using ℓ invocations of SAVSS-Share, one for each $s^l \in S$, D can share S with a private communication of $\mathcal{O}((\ell n^4 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(\ell n^3 \log n)$ bits, protocol SAVSS-MS-Share achieves the same task with a private communication of $\mathcal{O}((\ell n^3 + n^4 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^3 \log n)$ (independent of ℓ) bits. This shows that executing a *single instance* of SAVSS-MS dealing with *multiple secrets* concurrently is advantageous over executing *multiple instances* of SAVSS dealing with *single secret*.

The structure of SAVSS-MS-Share is divided into same three phases as in SAVSS-Share. The corresponding protocols are Commitment-MS, Verification-MS and Re-commitment-MS. They are simple extension of the corresponding protocols in SAVSS-Share and are presented in Fig. 8.13, Fig. 8.14 and Fig. 8.15.

Now protocol SAVSS-MS-Share$(D, \mathcal{P}, S, \epsilon)$ consists of the code presented in Commitment-MS$(D, \mathcal{P}, S, \epsilon)$, Verification-MS$(D, \mathcal{P}, S, \epsilon)$ and Re-commitment-MS$(D, \mathcal{P}, S, \epsilon)$ in this order. Protocol SAVSS-MS-Rec-Private$(D, \mathcal{P}, S, P_\alpha, \epsilon)$ and SAVSS-MS-Rec-Public$(D, \mathcal{P}, S, \epsilon)$ are very straight forward extension of SAVSS-Rec-Private and SAVSS-Rec-Public. The protocol SAVSS-MS is presented in Fig. 8.16. The proofs for the properties of the protocols dealing with multiple secrets will be similar to the proofs of the protocols dealing with single secret.

Theorem 8.43 *Protocol AVSS-MS consisting of (SAVSS-MS-Share, SAVSS-MS-Rec-Private,SAVSS-MS-Rec-Public) constitutes a valid strong statistical AVSS scheme for $\ell \geq 1$ secrets.*

Remark 8.44 (D's commitment in SAVSS-MS-Share) *We say that D has committed secret $S \in \mathbb{F}^\ell$ in SAVSS-MS-Share if there are ℓ degree-t univariate polynomials, $f^1(x), \ldots, f^\ell(x)$, such that $f^l(0) = s^l$ for $l = 1, \ldots, \ell$ and every honest P_i in $VCORE$ receives $(f^1(i), \ldots, f^\ell(i))$ from D and commits to $(f^1(i), \ldots, f^\ell(i))$ using AWSS-MS-Share. In protocol SAVSS-MS-Share, $f^l(x) = f^l_0(x) = \overline{F^l}(x, 0)$ for every $l = 1, \ldots, \ell$, where $\overline{F^1}(x, y), \ldots, \overline{F^\ell}(x, y)$ are D's committed bivariate polynomial. When D is honest, $\overline{F^l}(x, y) = F^l(x, y)$ for $l = 1, \ldots, \ell$.*

Figure 8.13: Code for Commitment by D Phase for $\ell \geq 1$ secrets

Code Commitment-MS($D, \mathcal{P}, S, \epsilon$)

i. Distribution by D: Code for D — Only D executes this code

1. Select ℓ random degree-(t, t) bivariate polynomials $F^1(x, y), \ldots, F^\ell(x, y)$ such that $F^l(0, 0) = s^l$ for $l = 1, \ldots, \ell$.
2. Send $f_i^l(x) = F^l(x, i)$ and $g_i^l(y) = F^l(i, y)$ for $l = 1, \ldots, \ell$ to P_i, for $i = 1, \ldots, n$.
3. For $i = 1, \ldots, n$, initiate AWSS-MS-Share$(D, \mathcal{P}, (f_i^1(x), \ldots, f_i^\ell(x)), \epsilon')$ (see Notation 8.14 for the syntax) for sharing $(f_i^1(x), \ldots, f_i^\ell(x))$, where $\epsilon' = \frac{\epsilon}{n}$.

ii. Code for P_i: – Every party in $\mathcal{P}$, including D, executes this code

1. Wait to receive degree-t polynomials $f_i^l(x)$ and $g_i^l(y)$ for $l = 1, \ldots, \ell$ from D.
2. Participate in AWSS-MS-Share$(D, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), \epsilon')$ by executing steps in [**Verification: Code for** P_i] (of AWSS-MS-Share) for all $j = 1, \ldots, n$.
3. After the completion of step 1 of [**Verification: Code for** P_i] for all the n invocations of AWSS-MS-Share, check whether $g_i^l(j) = f_j^l(i)$ holds for all $j = 1, \ldots, n$ and $l = 1, \ldots, \ell$, where $f_j^l(i)$ is obtained from the execution of AWSS-MS-Share$(D, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), \epsilon')$. If yes then A-cast `Matched-Column`.

Rest of the steps are same as in Commitment.

Notation 8.45 (Notation for SAVSS-MS-Share and SAVSS-MS-Rec-Private) *Later we will invoke SAVSS-MS-Share as SAVSS-MS-Share $(D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x)), \epsilon)$ to mean that D commits $f^1(x), \ldots, f^\ell(x)$ in SAVSS-MS-Share. Essentially here D is asked to choose bivariate polynomials $F^1(x, y), \ldots, F^\ell(x, y)$, each of degree-$(t, t)$ such that $F^l(x, 0) = f^l(x)$ holds for $l = 1, \ldots, \ell$. Similarly, SAVSS-MS-Rec-Private will be invoked as SAVSS-MS-Rec-Private$(D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x)), P_\alpha, \epsilon)$ to enable P_α-private-reconstruction of $(f^1(x), \ldots, f^\ell(x))$.*

8.5.4 Deciding The Choice of AWSS Protocol

For our AMPC protocol presented in Chapter 10, we require a strong statistical AVSS with P_α-private-reconstruction. We now examine which AWSS will fit better in this case in terms of communication complexity. For this we now analyze the communication complexity of our SAVSS-MS protocol by substituting AWSS-MS-I and AWSS-MS-II.

Communication complexity of SAVSS-MS using AWSS-MS-I as a Black box: Protocol SAVSS-MS-Share incurs a private communication of $\mathcal{O}((\ell n^3 +$

Figure 8.14: Code for Verification of D's Commitment Phase for $\ell \geq 1$ secrets

Code Verification-MS($D, \mathcal{P}, S, \epsilon$)

i. P_j-Weak-Private-Reconstruction of $(f_j^1(x), \ldots, f_j^\ell(x))$ for $j = 1, \ldots, n$:
CODE FOR P_i — Every party in $\mathcal{P}$ executes this code.

1. After agreeing on $WCORE$ and corresponding OKP_j's, participate in AWSS-MS-Rec-Private$(D, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), P_j, \epsilon')$, for $j = 1, \ldots, n$, to enable P_j-weak-private-reconstruction of $(f_j^1(x), \ldots, f_j^\ell(x))$. Notice that same $WCORE$ is used in each AWSS-SS-Rec-Private$(D, \mathcal{P}, f_j(x), P_j, \epsilon')$, for $j = 1, \ldots, n$
2. At the completion of AWSS-MS-Rec-Private$(D, \mathcal{P}, (f_i^1(x), \ldots, f_i^\ell(x)), P_i, \epsilon')$, obtain either degree-$t$ polynomials $\overline{f_i^1}(x), \ldots, \overline{f_i^\ell}(x)$ or $NULL$.
3. If $f_i^l(x) = \overline{f_i^l}(x)$ for all $l = 1, \ldots, \ell$, then A-cast `Matched-Row`.

Rest of the steps are same as in Verification.

Figure 8.15: Code for "Re-commitment by Individual Parties" Phase for $\ell \geq 1$ secrets

Code Re-commitment-MS($D, \mathcal{P}, S, \epsilon$)

i. Code for P_i: — Every party executes this code

1. If you have A-casted `Matched-Row` in Verification-MS then initiate AWSS-MS-Share$(P_i, \mathcal{P}, (f_i^1(x), \ldots, f_i^\ell(x), \epsilon')$ to re-commit $(f_i^1(x), \ldots, f_i^\ell(x))$, where $\epsilon' = \frac{\epsilon}{n}$.
2. For each j, such that P_j has A-casted `Matched-Row` in Verification-MS, participate in AWSS-MS-Share$(P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x), \epsilon')$ by executing steps in [**Verification: Code for** P_i] (of AWSS-MS-Share) in the following way:

   After the completion of step 1 of [**Verification: Code for** P_i], check whether $g_i^l(j) = f_j^l(i)$ for $l = 1, \ldots, \ell$ holds, where $(f_j^1(i), \ldots, f_j^\ell(i))$ are obtained from the execution of AWSS-SS-Share$(P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x), \epsilon')$ and $(g_i^1(y), \ldots, g_i^\ell(y))$ was obtained from D in code Commitment-MS. If yes then participate in the next steps in [**Verification: Code for** P_i] corresponding to AWSS-MS-Share$(P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x), \epsilon')$.

Rest of the steps are same as in Re-commitment except that at every place AWSS-Share$(P_i, \mathcal{P}, f_i(x), \epsilon')$ is replaced by AWSS-MS-Share$(P_i, \mathcal{P}, (f_i^l(x), \ldots, f_i^\ell(x)), \epsilon')$.

Figure 8.16: Our Strong Statistical AVSS for Sharing $\ell \geq 1$ Secrets with $n = 3t + 1$

Protocol SAVSS-MS($D, \mathcal{P}, S, \epsilon$)

SAVSS-MS-Share($D, \mathcal{P}, S, \epsilon$)

1. Replicate Code Commitment-MS($D, \mathcal{P}, S, \epsilon$).
2. Replicate Code Verification-MS($D, \mathcal{P}, S, \epsilon$).
3. replicate Code Re-commitment-MS($D, \mathcal{P}, S, \epsilon$).

SAVSS-MS-Rec-Private($D, \mathcal{P}, S, P_\alpha, \epsilon$): P_α-private-reconstruction of S:

P_α-weak-private-reconstruction of $(f_j^1(x), \ldots, f_j^\ell(x))$ for every $P_j \in VCORE$: Code for P_i

1. Participate in AWSS-MS-Rec-Private($P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), P_\alpha, \epsilon'$) for every $P_j \in VCORE$, where $\epsilon' = \frac{\epsilon}{n}$.

Local Computation: Code for P_α

1. For every $P_j \in VCORE$, obtain either $(\overline{f_j^1}(x), \ldots, \overline{f_j^\ell}(x))$ or $NULL$ from P_α-weak-private-reconstruction. Add party $P_j \in VCORE$ to REC if non-NULL output is obtained.
2. Wait until $|REC| = t + 1$. Construct bivariate polynomial $\overline{F^1}(x, y), \ldots, \overline{F^\ell}(x, y)$ such that $\overline{F^l}(x, j) = \overline{f_j^l}(x)$ for every $P_j \in REC$ and every $l = 1, \ldots, \ell$. Compute $\overline{s^l} = \overline{F^l}(0, 0)$ for every $l = 1, \ldots, \ell$ and terminate SAVSS-MS-Rec-Private.

SAVSS-Rec-Public($D, \mathcal{P}, S, \epsilon$): Public reconstruction of S:

Public reconstruction of $(f_j^1(x), \ldots, f_j^\ell(x))$ for every $P_j \in VCORE$: Code for P_i

1. Participate in AWSS-MS-Rec-Public($P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), \epsilon'$) for every $P_j \in VCORE$. We denote AWSS-Rec-Public($P_j, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), \epsilon'$) by AWSS-Rec-Public$_j$

Local Computation: Code for P_i
Same code as presented above for P_α in SAVSS-MS-Rec-Private.

$n^4 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^3 \log n)$ bits (as it requires $\mathcal{O}(n)$ executions of AWSS-MS-Share and AWSS-MS-Rec-Private). Protocol SAVSS-MS-Rec-Private incurs private communication of $\mathcal{O}((\ell n^3 + n^4 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits (as it requires $\mathcal{O}(n)$ executions of AWSS-MS-Rec-Private). Protocol SAVSS-MS-Rec-Public incurs A-cast communication of $\mathcal{O}((\ell n^3 + n^4 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits (as it requires $\mathcal{O}(n)$ executions of AWSS-MS-Rec-Public).

Communication complexity of SAVSS-MS using AWSS-MS-II as a Black

box: Protocol SAVSS-MS-Share incurs a private communication of $\mathcal{O}((\ell n^3 + n^4)\log\frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}((\ell n^3 + n^4)\log\frac{1}{\epsilon})$ bits. Protocol SAVSS-MS-Rec-Private incurs private communication of $\mathcal{O}((\ell n^3+n^4)\log\frac{1}{\epsilon})$ bits. Protocol SAVSS-MS-Rec-Public incurs A-cast communication of $\mathcal{O}((\ell n^3 + n^4)\log\frac{1}{\epsilon})$ bits.

So if we consider SAVSS-MS with P_α-private-reconstruction i.e (SAVSS-MS-Share,SAVSS-MS-Rec-Private) then the total communication is better when AWSS-MS-I is used as black box (which is private communication of $\mathcal{O}((\ell n^3+n^4\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^3\log n)$ bits). Now notice that the A-cast communication is independent of ℓ in this case (which is required for our AMPC in order to maintain its efficiency by certain bound). So we will consider AWSS-MS-I as a black box for SAVSS-MS and state the communication complexity of SAVSS-MS in the following theorem. Before that we fix the field $\mathbb{F}$ over which SAVSS-MS should work to bound the error probability by ϵ.

We now fix the field $\mathbb{F}$ over which SAVSS-MS should work to bound the error probability by ϵ. To bound the error probability by ϵ, the computation in SAVSS-MS is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^4\kappa 2^{-\kappa}$. This is derived from the fact that in SAVSS-MS, AWSS-MS-I is invoked with $\frac{\epsilon}{n}$ error probability and as mentioned in subsection 8.3.4, $\epsilon \geq n^3\kappa 2^{-\kappa}$ should hold to bound error probability of AWSS-MS-I by ϵ. So here each element from the field is represented by $\kappa = \log|\mathbb{F}| = \mathcal{O}(\log\frac{1}{\epsilon})$ bits.

Theorem 8.46 (Communication Complexity of SAVSS-MS) *Using AWSS-MS-I as building block, the communication complexity of SAVSS-MS becomes as follows:*

- *Protocol SAVSS-MS-Share incurs a private communication of $\mathcal{O}((\ell n^3+n^4\log\frac{1}{\epsilon})$ $\log\frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^3\log n)$ bits.*
- *Protocol SAVSS-MS-Rec-Private private communication of $\mathcal{O}((\ell n^3+n^4\log\frac{1}{\epsilon})$ $\log\frac{1}{\epsilon})$ bits.*

PROOF: The proof follows from Lemma 8.17 and the fact that SAVSS-MS-Share invokes $\Theta(n)$ instances of AWSS-MS-Share and AWSS-MS-Rec-Private, each executed with an error probability of $\epsilon' = \frac{\epsilon}{n}$. Moreover, SAVSS-MS-Rec-Private invokes $\Theta(n)$ instances of AWSS-MS-Rec-Private. □

8.6 Conclusion and Open Questions

In this chapter, we presented two novel statistical AVSS protocols with *optimal resilience*; i.e. with $n = 3t + 1$; one protocol for weak statistical AVSS and the other for strong statistical AVSS. The weak statistical AVSS with its *public reconstruction* will be used in our ABA protocol (in Chapter 9). On the other hand, the strong statistical AVSS with its *private reconstruction* will be used in our AMPC protocol (in Chapter 10). Our strong statistical AVSS protocol enjoys the following properties:

1. The strong statistical AVSS makes sure that the shared value(s) always belong to $\mathbb{F}$ even when D is corrupted.
2. The A-cast communication of our strong AVSS is *independent* of the number of secrets i.e. ℓ;

But as evident from the discussions, our strong statistical AVSS is much more complicated than weak statistical AVSS. Nevertheless, both the AVSS protocols show significant improvement over the only known statistical AVSS of [39] in terms of the communication complexity. The protocols are also based on completely disjoint techniques. We conclude this chapter with the following open question:

Open Problem 13 *Can we design weak and strong statistical AVSS with lesser communication complexity than that is reported in this chapter?*

More generally, we may ask the following question.

Open Problem 14 *What is the lower bound on the communication complexity of statistical AVSS protocols with optimal resilience?*

8.7 Appendix: Analysis of the Communication Complexity of the AVSS Scheme of [39]

The communication complexity analysis of the AVSS protocol of [39] was not reported anywhere so far. So we have carried out the same at this juncture. To do so, we have considered the detailed description of the AVSS protocol of [39] given in Canetti's Thesis [35]. In [35], the AVSS is designed with public reconstruction. To bound the error probability by ϵ, all the communication and computation in the protocol of [39] is done over a finite field $\mathbb{F}$, where $|\mathbb{F}| = GF(2^\kappa)$ and $\epsilon = 2^{-\Omega(\kappa)}$. Thus each field element can be represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

To begin with, in the AICP protocol of [39], D gives $\mathcal{O}(\kappa)$ field elements to INT and $\mathcal{O}(\kappa)$ field elements to verifier R. Though the AICP protocol of [35] is presented with a *single* verifier, it is executed with n verifiers in protocol A-RS. In order to execute AICP with n verifiers, D gives $\mathcal{O}(n\kappa)$ field elements to INT and $\mathcal{O}(\kappa)$ field elements to each of the n verifiers. So the communication complexity of AICP of [35] when executed with n verifiers is $\mathcal{O}(n\kappa)$ field elements and hence $\mathcal{O}(n\kappa^2)$ bits.

Now by incorporating their AICP protocol with n verifiers in Shamir secret sharing [140], the authors in [39] designed an asynchronous primitive called A-RS, which consists of two sub-protocols, namely A-RS-Share and A-RS-Rec. In the A-RS-Share protocol, D generates n shares (Shamir shares) of a secret s and for each of the n shares, D executes an instance of AICP protocol with n verifiers. So the A-RS-Share protocol of [39] involves a private communication of $\mathcal{O}(n^2\kappa^2)$ bits. In addition to this, the A-RS-Share protocol also involves an A-cast of $\mathcal{O}(\log(n))$ bits. In the A-RS-Rec protocol, the IC signatures given by D in A-RS-Share are revealed, which involves a private communication of $\mathcal{O}(n^2\kappa^2)$ bits. In addition, the A-RS-Rec protocol involves A-cast of $\mathcal{O}(n^2 \log(n))$ bits.

Proceeding further, by incorporating their A-RS protocol, the authors in [39] designed an AWSS scheme. The AWSS protocol consists of two sub-protocols, namely AWSS-Share and AWSS-Rec. In the AWSS-Share protocol, D generates n shares (Shamir shares [140]) of the secret and instantiate n instances of the AICP protocol for each of the n shares. Now each individual party A-RS-Shares all the values that it has received in the n instances of the AICP protocol. Since each individual party receives a total of $\mathcal{O}(n\kappa)$ field elements in the n instances of AICP, the above step incurs a private communication of $\mathcal{O}(n^4\kappa^3)$ bits and

A-cast of $\mathcal{O}(n^2\kappa\log(n))$ bits. In the AWSS-Rec protocol, each party P_i tries to reconstruct the values which are A-RS-Shared by each party P_j in a set $\mathcal{E}_i$. Here $\mathcal{E}_i$ is a set which is defined in the AWSS-Share protocol. In the worst case, the size of each $\mathcal{E}_i$ is $\mathcal{O}(n)$. So in the worst case, the AWSS-Rec protocol privately communicates $\mathcal{O}(n^5\kappa^3)$ bits and A-casts $\mathcal{O}(n^5\kappa\log(n))$ bits.

The authors in [39] then further extended their AWSS-Share protocol to Two& Sum AWSS-Share protocol, where each party P_i has to A-RS-Share $\mathcal{O}(n\kappa^2)$ field elements. So the communication complexity of Two& Sum AWSS-Share is $\mathcal{O}(n^4\kappa^4)$ bits and A-cast of $\mathcal{O}(n^2\kappa^2\log(n))$ bits.

Finally using their Two&Sum AWSS-Share and AWSS-Rec protocol, the authors in [39] have deigned their AVSS scheme, which consists of two sub-protocols, namely AVSS-Share and AVSS-Rec. In the AVSS-Share protocol, the most communication expensive step is the one where each party has to AWSS-Rec $\mathcal{O}(n^3\kappa)$ field elements. So in total, the AVSS-Share protocol of [39] involves a communication complexity of $\mathcal{O}(n^9\kappa^4)$ bits and A-cast of $\mathcal{O}(n^9\kappa^2\log(n))$ bits. The AVSS-Rec protocol involves n instances of AWSS-Rec, resulting in a communication complexity of $\mathcal{O}(n^6\kappa^3)$ bits and A-cast of $\mathcal{O}(n^6\kappa\log(n))$ bits. As mentioned earlier, $\kappa = \mathcal{O}(\log\frac{1}{\epsilon})$. Replacing the value of κ, we obtain the following:

- AVSS-Share protocol of [39] requires a communication complexity of $\mathcal{O}(n^9(\log\frac{1}{\epsilon})^4)$ bits and A-cast of $\mathcal{O}(n^9(\log\frac{1}{\epsilon})^2\log(n))$ bits.

- AVSS-Rec protocol requires a communication complexity of $\mathcal{O}(n^6(\log\frac{1}{\epsilon})^3)$ bits and A-cast of $\mathcal{O}(n^6(\log\frac{1}{\epsilon})\log(n))$ bits.

Chapter 9

Efficient ABA with Optimal Resilience for Short Message

An important variant of BA is Asynchronous Byzantine Agreement (ABA). An ABA protocol is carried out among n parties in a completely asynchronous network, where every two parties are directly connected by a secure channel and t out of the n parties are under the control of a *computationally unbounded Byzantine (active)* adversary $\mathcal{A}_t$. The communication complexity of ABA is one of its most important complexity measures. In this chapter, we present an efficient ABA protocol whose communication complexity is significantly better than the communication complexity of the existing ABA protocols in the literature. Our protocol is *optimally resilient* and thus requires $n = 3t+1$ parties for its execution.

Specifically, the *amortized* communication complexity of our ABA protocol is $\mathcal{O}(\mathcal{C}n^4 \log \frac{1}{\epsilon})$ bits for attaining agreement on a *single* bit, where ϵ denotes the probability of non-termination and $\mathcal{C}$ denotes the *expected running time* of our protocol. Conditioned on the event that our ABA protocol terminates, it does so in constant expected time; i.e., $\mathcal{C} = \mathcal{O}(1)$. We compare our result with most recent optimally resilient, ABA protocols proposed in [39] and [1] and show that our protocol gains by a factor of $\mathcal{O}(n^7 (\log \frac{1}{\epsilon})^3)$ over the ABA of [39] and by a factor of $\mathcal{O}(n^4 \frac{\log n}{\log \frac{1}{\epsilon}})$ over the ABA of [1].

As a key tool, we use the weak statistical AVSS designed in Chapter 8. The common coin primitive is one of the most important building blocks for the construction of ABA protocol. The only known efficient (i.e polynomial communication complexity) common coin protocol [67, 35] uses AVSS sharing a single secret as a black-box. Unfortunately, the known common coin protocol does not achieve its goal when multiple invocations of AVSS sharing single secret are replaced by single invocation of AVSS sharing multiple secrets. Hence in this chapter, we twist the existing common coin protocol to make it compatible with our new AVSS that can share multiple secrets concurrently. As a byproduct, our new common coin protocol is much more communication efficient than the existing common coin protocol.

9.1 Introduction

The problem of Byzantine Agreement (BA) was introduced in [132] and since then it has emerged as the most fundamental problem in distributed computing. It has been used as building block for several important secure distributed computing tasks such as MPC [151, 95, 20, 138, 48, 98, 12, 111, 52, 14], VSS [43, 138, 91, 73, 109] etc. The BA problem has been investigated extensively in various models, characterized by the synchrony of the network, privacy of the channels, computational power of the faulty parties and many other parameters [68, 18, 29, 39, 35, 118, 72, 110, 2, 24, 25, 26, 30, 31, 32, 44, 56, 54, 57, 59, 60, 61, 74, 70, 71, 65, 67, 78, 86, 89, 114, 117, 134, 136, 150, 148, 149]. An interesting and practically motivated variant of BA is ABA tolerating a *computationally unbounded* malicious adversary. This problem has got relatively less attention in comparison to the BA problem in synchronous network (see [118, 72] and their references). Since asynchronous networks model the real life networks like Internet more appropriately than synchronous networks, the fundamental problem like BA is worthy of deep investigation over asynchronous networks.

9.1.1 The Network and Adversary Model

This is same as described in section 8.1.1. Recall that the set of parties is denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$ and t out of the n parties can be under the influence of a *computationally unbounded Byzantine (active) adversary*, denoted as $\mathcal{A}_t$. We emphasize that we use $n = 3t + 1$ in this chapter.

9.1.2 Definitions

We now formally define ABA. An ABA protocol allows the set of n parties in $\mathcal{P}$, each having a private input binary value, to agree on a consensus value, despite the presence of $\mathcal{A}_t$.

Definition 9.1 (ABA [39]) *: Let* Π *be an asynchronous protocol executed among the set of parties* $\mathcal{P}$*, with each party having a private binary input. We say that* Π *is an ABA protocol tolerating* $\mathcal{A}_t$ *if the following hold, for every possible behavior of* $\mathcal{A}_t$ *and every possible input:*

1. **Termination***: All honest parties eventually terminate the protocol.*
2. **Correctness***: All honest parties who have terminated the protocol hold identical outputs. Furthermore, if all honest parties had the same input, say* ρ*, then all honest parties output* ρ*.*

Remark 9.2 *The* **Termination** *property of* A-cast *(described in Chapter 7) is weaker than that of* ABA*. In* A-cast*, it is not required that the honest parties terminate the protocol if* S *is faulty; but in ABA, honest parties are required to terminate the protocol always.*

We now define (ϵ, δ)-ABA protocol, where both ϵ and δ are negligibly small values (Recall the discussion presented in the beginning of section 1.5 for the meaning of negligible) and are called as error probabilities of the ABA protocol. Moreover, we have $n = \mathcal{O}(\log \frac{1}{\epsilon})$ and $n = \mathcal{O}(\log \frac{1}{\delta})$ (follows from the definition of negligible, i.e $\epsilon \leq \frac{1}{2^{\alpha n}}$ and $\delta \leq \frac{1}{2^{\alpha n}}$ as mentioned in section 1.5).

Definition 9.3 ((ϵ, δ)-ABA) *: An ABA protocol* Π *is called* (ϵ, δ)*-ABA if*

1. Π *satisfies* **Termination** *described in Definition 9.1, except with an error probability of* ϵ *and*

2. *Conditioned on the event that every honest party terminates* Π*, protocol* Π *satisfies* **Correctness** *property described in Definition 9.1, except with error probability* δ*.*

9.1.3 Relevant History of ABA

From [132, 118], any ABA protocol tolerating $\mathcal{A}_t$ is possible iff $n \geq 3t+1$. Thus any ABA protocol designed with $n = 3t+1$ is therefore called as *optimally resilient*. By the seminal result of [71], any ABA protocol, irrespective of the value of n, must have some *non-terminating* runs/executions, where some honest party(ies) may not output any value and thus may not terminate at all. So in any (ϵ, δ)-ABA protocol with non-zero ϵ, the probability of the occurrence of a non-terminating execution is at most ϵ (these type of protocols are called $(1-\epsilon)$-terminating [39, 35]). On the other hand in any $(0, \delta)$-ABA protocol, the *probability* of occurrence of a non-terminating execution is *asymptotically zero* (these type of protocols are called *almost-surely terminating*, a term coined by Abraham et al. in [1]).

We now describe the chain of results that has appeared in the literature of ABA. Rabin [136] and Ben-Or [18] presented ABA protocols with $n \geq 8t+1$ and $n \geq 5t+1$ respectively. Since, both these protocols were not *optimally resilient*, researchers have tried to design ABA protocol with *optimal resilience* or close to optimal resilience. In this direction the first attempt is by Bracha [29] who reported an optimally resilient $(0,0)$-ABA protocol. However, the protocol of Bracha [29] requires exponential ($\Theta(2^n)$) expected time and exponential ($\Theta(2^n)$) communication complexity. Subsequently, Feldman and Micali [66] presented a $(0,0)$-ABA protocol which runs in constant expected time and requires polynomial communication complexity (they actually extend their BA protocol in synchronous settings [66] to asynchronous settings). However, the ABA protocol of Feldman and Micali [66] is not optimally resilient and requires $4t+1$ parties. So it remained an open question whether there exists an optimally resilient ABA with polynomial running time and communication complexity. Canetti and Rabin [39] answered this question in affirmative and provided an $(\epsilon, 0)$-ABA protocol that offers optimal resilience, *constant* expected running time and polynomial communication complexity. But it is to be noted that so far in the literature there was no optimally resilient $(0,0)$-ABA protocol with polynomial communication complexity. This long standing open question was resolved by Abraham et al. [1]. However, the protocol of [1] requires polynomial running time (as opposed to constant expected running time achieved by the ABA protocols of [66, 39]). Hence indeed there is an interesting open problem to come up with an optimally resilient $(0,0)$-ABA with constant expected running time and polynomial communication complexity. In Table 9.1, we summarize the best known existing ABA protocols.

Over a period of time, the techniques and the design approaches of ABA has evolved spectacularly. In his seminal paper [136], Rabin reduced the problem of ABA to that of a 'common coin'. Specifically, Rabin designed an ABA assuming that the parties have access to a 'common coin' (namely, a common source of randomness). However Rabin did not provide any implementation of common coin. In brief, common coin protocol allows the honest parties to output a common random bit with some probability which we may call as success probability of that common coin protocol. The first ever implementation of common coin was

Table 9.1: Summary of Best Known Existing ABA Protocols

Ref.	Type	Resilience	Communication Complexity (CC) in bits	Expected Running Time (ERT)
[29]	$(0,0)$-ABA	$t < n/3$	$\mathcal{O}(2^n)$	$\mathcal{C} = \mathcal{O}(2^n)$
[66, 67]	$(0,0)$-ABA	$t < n/4$	Private[a]: $\mathcal{O}(n^4\kappa \log \lvert\mathbb{F}\rvert)$[b] A-cast[c]: $\mathcal{O}(n^4\kappa \log \lvert\mathbb{F}\rvert)$	$\mathcal{C} = \mathcal{O}(1)$
[39, 35]	$(\epsilon,0)$-ABA	$t < n/3$	Private: $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^4)$ A-cast: $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^2 \log n)$	$\mathcal{C} = \mathcal{O}(1)$
[1]	$(0,0)$-ABA	$t < n/3$	Private: $\mathcal{O}(\mathcal{C}n^6 \log \lvert\mathbb{F}\rvert)$ A-cast: $\mathcal{O}(\mathcal{C}n^6 \log \lvert\mathbb{F}\rvert)$	$\mathcal{C} = \mathcal{O}(n^2)$

[a] Communication over private channels between pair of parties in $\mathcal{P}$.
[b] Here $\mathbb{F}$ is the finite field over which the ABA protocol of [66, 67] works. It is enough to have $|\mathbb{F}| \geq n$ and therefore $\log |\mathbb{F}|$ can be replaced by $\log n$. In fact in the remaining table, $\mathbb{F}$ bears the same meaning. Also here κ is the error parameter of the protocols.
[c] Total number of bits that needs to be A-casted.

done by Bracha [29]. However, the common coin protocol of [29] is very straight forward. Essentially in Bracha's common coin protocol every party tosses a coin locally and then they hope that they all got the same value; clearly this happens with probability which is exponentially small in the number of parties, namely $\Theta(2^{-n})$ (so the success probability of Bracha's common coin is $\Theta(2^{-n})$). Consequently the common coin of Bracha [29] while incorporated to design ABA causes the ABA of [29] to run for exponential expected time and also calls for exponential communication complexity. Bracha's design approach of ABA using common coin protocol provided an insightful implication which actually paved the future path for designing efficient ABA protocol with constant expected running time: *The expected running time of ABA is inversely proportional to the success probability of common coin protocol.* The above finding shows a natural direction towards designing efficient common coin protocol with constant success probability in order to construct an efficient ABA with constant expected running time (following the design approach of Bracha).

That is what is exactly achieved by Feldman and Micali [66, 67], who are the first to come up with a common coin protocol that has constant success probability in contrast to the exponential success probability of Bracha [29]. Using the new common coin, the ABA of [66, 67] follows the same design approach of Bracha and achieves constant expected running time and polynomial communication complexity. At the heart of the common coin protocol of [66] is an efficient AVSS protocol. In fact, the essence of [66] is the reduction of the common coin to that of implementing an AVSS protocol. Given an AVSS with n parties, the common coin of [66] requires $\min(n, 3t+1)$ parties for execution. In [67], Feldman et al. have designed an AVSS with $n = 4t + 1$ parties and using the AVSS, they designed an ABA with $4t + 1$ parties.

After that the researchers almost followed the same approach of reducing the design of ABA to that of designing AVSS. To design an ABA with optimal resilience i.e $n = 3t+1$, Canetti et al. [39] have designed an AVSS with $n = 3t+1$ for the first time in literature. As the AVSS had negligible error probability in termination, the resultant ABA of [39] is of type $(\epsilon, 0)$ (in contrast to the ABA of [29, 66] which are of type $(0, 0)$). Recently, Abraham et al. [1] have reported a weaker variant of AVSS, named as shunning AVSS which was used to design shunning common coin which is further used to design ABA protocol. The shunning AVSS has no error probability in termination and thus the resultant ABA of [1] is of type $(0, 0)$. But the shunning AVSS satisfies all the properties of AVSS only when all the parties including the corrupted ones behave according to the protocol steps. On the other hand, when at least a single corrupted party misbehaves, the shunning AVSS ensures at least one *honest* party will shun a *corrupted* party from then onwards. Due to this property of shunning AVSS, the ABA [1] protocol requires $\mathcal{O}(n^2)$ expected running time (in contrast to constant running time of the ABA protocol of [66, 39]). A more detailed discussion on the ABA protocols of [39] and [1] is presented later.

9.1.4 The Motivation of Our Work

The communication complexity of BA protocol is one of its important parameters. In the literature, a lot of attention has peen paid to improve the communication complexity of BA protocols in synchronous settings (see for example [26, 44, 57, 134, 75]). Unfortunately, not too much attention has been paid to design communication efficient ABA protocols with optimal resilience. Though the communication complexity of the known optimally resilient ABA protocols [39, 1] is polynomial in n and $\log \frac{1}{\epsilon}$, they involve fairly very high communication complexity. Especially, though the AVSS (and hence ABA) protocol of [39] is a seminal result, it is very much involved and complex in nature. In a real-life distributed network, fast and communication efficient protocols find lot of application. Naturally, designing optimally resilient, communication efficient, fast ABA protocol which runs in constant expected time is an important and interesting problem. Our result in this chapter marks a significant progress in this direction. Above all our ABA protocol is reasonably simple.

9.1.5 Contribution of This Chapter

In this chapter, we present an optimally resilient, $(\epsilon, 0)$-ABA protocol whose *amortized* communication complexity for agreeing on a single bit is $\mathcal{O}(\mathcal{C}n^4 \log \frac{1}{\epsilon})$ bits of private communication as well as $\mathsf{A\text{-}cast}$, where $\mathcal{C}$ is the *expected running time* of the protocol. Specifically, our ABA protocol requires private communication, as well as $\mathsf{A\text{-}cast}$ of $\mathcal{O}(\mathcal{C}n^5 \log \frac{1}{\epsilon})$ bits for reaching agreement on $t+1 = \Theta(n)$ bits *concurrently*. Conditioned on the event that our ABA protocol terminates, it does so in constant expected time; i.e., $\mathcal{C} = \mathcal{O}(1)$.

We compare our ABA with the optimally resilient $(\epsilon, 0)$-ABA protocol of [39] which also has constant expected running time; i.e., $\mathcal{C} = \mathcal{O}(1)$. The ABA of [39] *privately* communicates $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^4)$ bits and $\mathsf{A\text{-}casts}$ $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^2 \log n)$ bits. So our ABA achieves a huge gain in communication complexity over the ABA of [39], while keeping all other properties in place.

In another landmark work, Abraham et al. [1] proposed an optimally resilient $(0, 0)$-ABA protocol which requires $\mathcal{O}(\mathcal{C}n^6 \log n)$ bits of private communication as well as $\mathsf{A\text{-}cast}$. But ABA protocol of Abraham et al. takes polynomial ($\mathcal{C} = \mathcal{O}(n^2)$) expected time to terminate. Our ABA enjoys the following merits over the ABA of Abraham et al. [1]:

1. Our ABA is better in terms of communication complexity when $(\log \frac{1}{\epsilon}) < n^4 \log n$.

2. Our ABA runs in constant expected time. However, we stress that our ABA is of type $(\epsilon, 0)$ whereas ABA of [1] is of type $(0, 0)$.

In Table 9.2, we compare and contrast our ABA protocol with the ABA protocols of [39, 1].

Table 9.2: Comparison of Our ABA with Best Known Optimally Resilient ABA Protocols

Ref.	Type	Resilience	Communication Complexity (CC)	Expected Running Time (ERT)
[39]	$(\epsilon, 0)$	$t < \frac{n}{3}$	Private– $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^4)$ A-cast– $\mathcal{O}(\mathcal{C}n^{11}(\log \frac{1}{\epsilon})^2 \log n)$	$\mathcal{C} = \mathcal{O}(1)$
[1]	$(0, 0)$	$t < \frac{n}{3}$	Private– $\mathcal{O}(\mathcal{C}n^6 \log n)$ A-cast– $\mathcal{O}(\mathcal{C}n^6 \log n)$	$\mathcal{C} = \mathcal{O}(n^2)$
This Chapter†	$(\epsilon, 0)$	$t < \frac{n}{3}$	Private– $\mathcal{O}(\mathcal{C}n^4(\log \frac{1}{\epsilon}))$ A-cast– $\mathcal{O}(\mathcal{C}n^4(\log \frac{1}{\epsilon}))$	$\mathcal{C} = \mathcal{O}(1)$

† The communication complexity mentioned in the table is the amortized communication complexity of reaching agreement on a single bit message.

Our construction of ABA protocol employs the weak statistical AVSS scheme with $n = 3t + 1$ presented in Chapter 8. Our AVSS shares multiple secrets *concurrently* and brings forth several advantages of concurrently sharing multiple secrets.

As discussed earlier in subsection 9.1.3, the *common-coin* protocol is a very important building block of ABA protocol. Previously, AVSS sharing single secret was used to design the only known *common-coin* protocol with polynomial communication complexity [67, 35]. Informally, in the common coin protocol of [67], each party P_i in $\mathcal{P}$ is asked to act as a dealer and share n random secrets using AVSS. For this P_i invokes n parallel instances of AVSS as a dealer to share n secrets in parallel. It is obvious that we can do better if P_i invokes *single* instance of AVSS, which shares n secrets *concurrently*. However, our detailed analysis of the existing common coin protocol shows that the above modification does not lead to a correct solution for common coin protocol. Hence we bring several new modifications to the existing *common-coin* protocol so that it can use our new AVSS (that can share multiple secrets concurrently). As a result, our new common coin protocol is more communication efficient than the existing common coin protocol of [35, 39].

Our ABA protocol has error probability of ϵ in **Termination**. To bound the error probability by ϵ, all our protocols work over a finite field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq 4n^6 2^{-\kappa}$. Each field element can be represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits (this can be derived using $n = \mathcal{O}(\log \frac{1}{\epsilon})$). In order to bound the error probability of our ABA protocol by some specific value of ϵ, we find out the *minimum* value of κ that satisfies the relation between κ and ϵ. The value for κ will consequently determine the field $\mathbb{F}$ over which the protocol should work.

9.1.6 The Road-map

In section 9.2, we briefly discuss about the approaches used in the ABA protocols of [39], [1] and this chapter. Then for the ease of presentation, we divide the presentation of this chapter into two parts. In the first part presented in section 9.3, our focus is on a simple and clean presentation of an ABA for single bit. For this we consider our statistical weak AVSS protocol for single secret presented in Chapter 8. By incorporating this AVSS into the existing common coin protocol [67, 35], we devise an ABA scheme which allows the parties to agree on a *single* bit and requires private communication as well as A-cast of $\mathcal{O}(n^6(\log \frac{1}{\epsilon}))$ bits.

In the second part presented in section 9.4, we use our weak statistical AVSS scheme for sharing *multiple* secrets concurrently. Considering this AVSS, we then show how the existing common coin protocol of [67, 35] fails to achieve its goal, if we replace multiple invocations of AVSS sharing single secret by single invocation of our AVSS sharing multiple secrets. Subsequently we demonstrate how to modify the common coin protocol of [67, 35] and present a new common coin protocol that uses our AVSS sharing multiple secrets concurrently. Finally, using this common coin protocol, we present our new ABA scheme whose *amortized* communication cost of reaching agreement on a *single* bit is $\mathcal{O}(n^4(\log \frac{1}{\epsilon}))$ bits of private as well as A-cast communication. We then conclude this chapter with conclusion and open problems. Lastly, since the exact communication complexity analysis of the ABA scheme of [39] was not done earlier, we carry out the same in section 9.6 for the sake of completeness.

9.2 A Brief Discussion on the Approaches Used in the ABA Protocols of [39, 1] and Current Chapter

We now briefly discuss the approach used in the optimally resilient ABA protocols of [39], [1] and the current chapter.

Approach of the ABA of [39, 35]: The ABA protocol of Canetti et al. [39, 35] uses the reduction from AVSS to ABA. Hence they have first designed an AVSS with $n = 3t + 1$. There are well known inherent difficulties in designing AVSS with $n = 3t+1$ (see [39, 35]). To overcome these difficulties, the authors in [39] used an approach to design their AVSS scheme which was discussed in section 8.2 of Chapter 8. Just to recall, pictorially, AVSS scheme of [39] is designed using the following route: *ICP* $\rightarrow$ *A-RS* $\rightarrow$ *AWSS* $\rightarrow$ *Two & Sum AWSS* $\rightarrow$ *AVSS*. Since the AVSS scheme is designed on top of so many sub-protocols, it becomes highly communication intensive as well as very much involved. The exact communication complexity analysis of the ABA scheme of [39] was not done earlier. For the sake of completeness, we carry out the same in section 9.6.

Approach of the ABA of [1]: The ABA protocol of [1] used the same reduction from AVSS to ABA as in [39], except that the use of AVSS is replaced by a variant of AVSS that the authors called *shunning* (asynchronous) VSS (SVSS), where each party is guaranteed to terminate *almost-surely.* SVSS is a slightly weaker notion of AVSS in the sense that if all the parties behave correctly, then SVSS satisfies all the properties of AVSS without any error. Otherwise it does not satisfy the properties of AVSS but enables some honest party to identify at least one corrupted party, whom the honest party shuns from then onwards. The use of SVSS instead of AVSS in generating common coin causes the ABA of [1] to run for $\mathcal{O}(n^2)$ expected time. The SVSS protocol requires private communication of $\mathcal{O}(n^4 \log(n))$ bits and A-cast of $\mathcal{O}(n^4 \log(n))$ bits.

Approach of the ABA of This chapter: Our ABA protocol also follows the same reduction from AVSS to ABA as in [39]. In the course of designing our ABA protocol, our first step is to design a communication efficient AVSS protocol. Instead of following a fairly complex route taken by [39], we follow a shorter and simpler route to design an AVSS scheme in Chapter 8: *ICP* $\rightarrow$ *AWSS* $\rightarrow$ *AVSS*. Beside this, we significantly improve each of these building blocks by employing new design approaches. Also each of the building blocks deals with multiple secrets concurrently and thus lead to significant gain in communication complexity.

As mentioned earlier, the existing common coin protocol [67, 35] calls AVSS dealing with single secret as a black box. Our detailed analysis of the existing common coin protocol shows that the common coin protocol does not achieve its properties when the invocations of AVSS sharing single secret are replaced by invocations of our AVSS sharing multiple secrets concurrently. Hence, we have modified the existing common coin protocol so that it can use our AVSS sharing multiple secrets as a building block. Together, this lead to our efficient ABA protocol which we believe to be much simpler than the ABA of [39].

9.3 Our ABA Protocol for Single Bit

In this section, we first design an ABA protocol for single bit using our weak statistical AVSS protocol for single secret, presented in subsections 8.4.1 and 8.4.2 in Chapter 8. For this, we first recall the existing common coin protocol with its AVSS instances replaced by our weak statistical AVSS protocol WAVSS consisting of sub-protocols (WAVSS-Share,WAVSS-Rec-Public). Then we recall and present the voting protocol from [35]. Finally, we recall the ABA protocol from [35] that uses the common coin and voting protocol as building blocks.

9.3.1 Existing Common Coin Protocol Using Our AVSS Protocol

Here we first recall the definition of common coin and then recall the construction of common coin protocol following the description of [35]. The common coin protocol invokes many instances of AVSS scheme. In the following description of our common coin protocol, we replace the AVSS scheme of [35] by our AVSS scheme WAVSS. We start with the definition of common coin protocol.

Definition 9.4 (Common Coin [35]) *Let π be an asynchronous protocol, where each party has local random input and binary output. We say that π is a $(1-\epsilon)$-terminating, t-resilient common coin protocol if the following requirements hold for every adversary $\mathcal{A}_t$:*

1. **Termination:** *With probability at least $(1-\epsilon)$, all honest parties terminate.*

2. **Correctness:** *For every value $\sigma \in \{0, 1\}$, with probability at least $\frac{1}{4}$ all honest parties output σ.*

The Intuition: The common coin protocol, referred as Common-Coin, consists of two stages. In the first stage, each party acts as a dealer and shares n random secrets, using n distinct instances of WAVSS-Share each with allowed error probability of $\epsilon' = \frac{\epsilon}{n^2}$. The i^{th} secret shared by *each* party is actually associated with party P_i. Once a party P_i terminates any $t+1$ instances of WAVSS-Share corresponding to the secrets associated with him, he A-casts the identity of the dealers of these secrets. We say that these $t+1$ secrets are `attached` to P_i and later these $t+1$ secrets will be used to compute a value that will be associated with P_i.

Now in the second stage, after terminating the WAVSS-Share instances of all the secrets attached to some P_i, party P_j is sure that a fixed (yet unknown) value is attached to P_i. Once P_j is assured that values have been attached to enough number of parties, he participates in WAVSS-Rec-Public instances of the relevant secrets. This process of ensuring that there are enough parties that are attached with values is the core idea of the protocol. Once all the relevant secrets are reconstructed, each party locally computes his binary output based on the reconstructed secrets, in a way described in the protocol presented in the sequel. Protocol Common-Coin now appears in Fig. 9.1.

To bound the error probability by ϵ, the computation of Common-Coin is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^6 2^{-\kappa}$. This is derived from the fact that in Common-Coin, WAVSS is invoked with $\frac{\epsilon}{n^2}$ error probability and as mentioned in subsection 8.4.2, $\epsilon \geq n^4 2^{-\kappa}$ should hold to bound error probability of WAVSS by ϵ. So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

Let E be an event, defined as follows: All invocations of protocol WAVSS have been terminated properly. That is, if an honest party has terminated WAVSS-Share, then a value, say s' is fixed. All honest parties will terminate the corresponding invocation of WAVSS-Rec-Public with output s'. Moreover if dealer D is honest then s' is D's shared secret. It is easy to see that event E occurs with probability at least $1 - n^2\epsilon' = 1 - \epsilon$.

We now state the following lemmas which are more or less identical to the Lemmas 5.28-5.31 presented in [35]. For the sake of completeness, we recall all of them with proofs.

Lemma 9.5 ([35]) *All honest parties terminate Protocol Common-Coin in constant time.*

PROOF: First we show that every *honest* party P_i will A-cast "`Reconstruct Enabled`" eventually. By the termination property of our AVSS protocol WAVSS (see Lemma 8.20), every honest party will eventually terminate all the instances of WAVSS-Share of every other honest party. As there are at least $n-t$ honest parties, for every honest party P_i, $\mathcal{T}_i$ will eventually contain at least $t+1$ parties (in fact $n-t$ parties) and thus P_i will eventually A-cast "`Attach` T_i `to` P_i". So eventually, P_i will receive "`Attach` T_j `to` P_j" from every *honest* P_j. Now since every party P_k that is included in $\mathcal{T}_j$ (of honest P_j) will be eventually included in $\mathcal{T}_i$ (follows from the termination property of WAVSS; see Lemma 8.20), $T_j \subseteq \mathcal{T}_i$

Figure 9.1: Existing Common Coin Protocol

Protocol Common-Coin($\mathcal{P}, \epsilon$)

Code for P_i: — Every party executes this code

1. For $j = 1, \ldots, n$, choose a random value x_{ij} and execute WAVSS-Share($P_i, \mathcal{P}, x_{ij}, \epsilon'$) where $\epsilon' = \frac{\epsilon}{n^2}$.
2. Participate in WAVSS-Share($P_j, \mathcal{P}, x_{jk}, \epsilon'$) for every $j, k \in \{1, \ldots, n\}$. We denote WAVSS-Share($P_j, \mathcal{P}, x_{jk}, \epsilon'$) by WAVSS-Share$_{jk}$.
3. Create a dynamic set $\mathcal{T}_i$. Add party P_j to $\mathcal{T}_i$ if WAVSS-Share($P_j, \mathcal{P}, x_{jk}, \epsilon'$) has been terminated for all $k = 1, \ldots, n$. Wait until $|\mathcal{T}_i| = t+1$. Then assign $T_i = \mathcal{T}_i$ and A-cast "`Attach` T_i `to` P_i". We say that the secrets $\{x_{ji} | P_j \in T_i\}$ are `attached` to party P_i.
4. Create a dynamic set $\mathcal{A}_i$. Add party P_j to $\mathcal{A}_i$ if
   (a) "`Attach` T_j `to` P_j" is received from the A-cast of P_j and
   (b) $T_j \subseteq \mathcal{T}_i$

   Wait until $|\mathcal{A}_i| = n - t$. Then assign $A_i = \mathcal{A}_i$ and A-cast "P_i `Accepts` A_i".
5. Create a dynamic set $\mathcal{S}_i$. Add party P_j to $\mathcal{S}_i$ if
   (a) "P_j `Accepts` A_j" is received from the A-cast of P_j and
   (b) $A_j \subseteq \mathcal{A}_i$.

   Wait until $|\mathcal{S}_i| = n - t$. Then A-cast "`Reconstruct Enabled`". Let H_i be the current content of $\mathcal{A}_i$.
6. Participate in WAVSS-Rec-Public($P_k, \mathcal{P}, x_{kj}, \epsilon'$) for every $P_k \in T_j$ of every $P_j \in \mathcal{A}_i$ (note that some parties may be included in $\mathcal{A}_i$ after the A-cast of "`Reconstruct Enabled`". The corresponding WAVSS-Rec-Public are invoked immediately). We denote WAVSS-Rec-Public($P_k, \mathcal{P}, x_{kj}, \epsilon'$) by WAVSS-Rec-Public$_{kj}$.
7. Let $u = \lceil 0.87n \rceil$. Every party $P_j \in \mathcal{A}_i$ is `associated` with a value, say V_j which is computed as follows: $V_j = (\sum_{P_k \in T_j} x_{kj}) \ \ mod \ \ u$ where x_{kj} is reconstructed back from WAVSS-Rec-Public($P_k, \mathcal{P}, x_{kj}, \epsilon'$) (if $NULL$ is reconstructed in some instance of WAVSS-Rec-Public then some predefined value $x^{\star}_{kj} \in \mathbb{F}$ will be taken as the secret; for details see Remark 8.24).
8. Wait until the values `associated` with all the parties in H_i are computed. Now if there exits a party $P_j \in H_i$ such that $V_j = 0$, then output 0. Otherwise output 1.

will hold good. Therefore, every honest P_j will be eventually included in $\mathcal{A}_i$. Thus for an honest P_i, $\mathcal{A}_i$ will eventually be of size $n - t$ and hence P_i will A-cast "P_i `Accepts` A_i". Now following the similar argument as above, we can show that for an honest P_i, $\mathcal{S}_i$ will eventually be of size $n-t$ and hence P_i will A-cast "Reconstruct Enabled".

Now it remains to show that WAVSS-Rec-Public protocols invoked by any honest party will be terminated eventually. Once this is proved, every honest party will terminate protocol Common-Coin after executing the remaining steps of Common-Coin such as computing V_i etc. By the previous argument given above, if an honest party P_i receives "Attach T_j to P_j" from P_j and includes P_j in $\mathcal{A}_i$, then eventually every other honest party will do the same. Hence if P_i invokes WAVSS-Rec-Public$_{kj}$ for $P_j \in \mathcal{A}_i$ and $P_k \in T_j$, then eventually every other honest party will also do the same. Now by the termination property of WAVSS (see Lemma 8.20), every WAVSS-Rec-Public$_{kj}$ protocols will be terminated by every honest party.

Given event E, all invocations of WAVSS-Share and WAVSS-Rec-Public terminate in constant time. The black box protocol for A-cast terminates in constant time. Thus protocol Common-Coin terminates in constant time. □

Lemma 9.6 ([35]) *In protocol Common-Coin, once some honest party P_j receives "**Attach** T_i **to** P_i" from the A-cast of P_i and includes P_i in $\mathcal{A}_j$, a unique value V_i is fixed such that*

1. *Every honest party will **associate** V_i with P_i, except with probability $1-\frac{\epsilon}{n}$.*
2. *V_i is distributed uniformly over $[0,\ldots,u]$ and is independent of the values **associated** with the other parties.*

PROOF: Once some honest party P_j receives "Attach T_i to P_i" from the A-cast of P_i and includes P_i in $\mathcal{A}_j$, a unique value V_i is fixed. Here $V_i = (\sum_{P_k \in T_i} x_{ki}) \ mod \ u$, where x_{ki} is value that is shared by P_k as a dealer during the execution of WAVSS-Share$_{ki}$. According to the protocol steps eventually all the honest parties will invoke WAVSS-Rec-Public$_{ki}$ corresponding to each $P_k \in T_i$ and consequently each honest party will reconstruct x_{ki} at the completion of WAVSS-Rec-Public$_{ki}$, except with probability ϵ' (recall that each instance of AVSS scheme has an associated error probability of ϵ'). Now since $|T_i| = t+1$, every honest party will associate V_i with P_i with probability at least $1-(t+1)\epsilon' \approx 1-\frac{\epsilon}{n}$.

Now it remains to show that V_i is uniformly distributed over $[0,\ldots,u]$ and is independent of the values associated with the other parties. An honest party starts reconstructing the secrets attached to P_i (i.e starts invoking WAVSS-Rec-Public$_{ki}$ for every $P_k \in T_i$) only after it receives "Attach T_i to P_i" from the A-cast of P_i. So the set T_i is fixed before any honest party invokes WAVSS-Rec-Public$_{ki}$ for some k. The secrecy property of WAVSS-Share ensures that corrupted parties will have no information about the value shared by any honest party until the value is reconstructed after executing corresponding WAVSS-Rec-Public. Thus when T_i is fixed, the values that are shared by corrupted parties corresponding to P_i are completely independent of the values shared by the honest parties corresponding to P_i. Now, each T_i contains at least one honest party and every honest party's shared secrets are uniformly distributed and mutually independent. Hence the sum V_i is uniformly and independently distributed over $[0,\ldots,u]$. □

Lemma 9.7 ([35]) *Once an honest party A-casts "**Reconstruct Enabled**", there exists a set M such that:*

1. *For every party $P_j \in M$, some honest party has received "**Attach** T_j **to** P_j" from the A-cast of P_j.*

2. *When any honest party P_j* A-casts *"Reconstruct Enabled", then it will hold that $M \subseteq H_j$.*

3. *$|M| \geq \frac{n}{3}$.*

PROOF: Let P_i be the *first honest party* to A-cast "Reconstruct Enabled". Then let $M = \{P_k \mid P_k$ belongs to A_l's of at least $t+1$ P_l's who belongs to $\mathcal{S}_i$ when P_i A-casted Reconstruct Enabled $\}$. We now show the parties in M satisfies the properties mentioned in the lemma.

It is clear that $M \subseteq H_i$. Thus party P_i has received "Attach T_j to P_j" from the A-cast of every $P_j \in M$. As P_i is assumed to be honest here, the first part of the lemma is asserted.

We now prove the second part. An *honest party* P_j A-casts "Reconstruct Enabled" only when $\mathcal{S}_j$ contains $n - t = 2t + 1$ parties. Now note that $P_k \in M$ implies that P_k belongs to A_l's of at least $t+1$ P_l's who belong to $\mathcal{S}_i$. This ensures that there is at least one such P_l who belongs to $\mathcal{S}_j$, as well as $\mathcal{S}_i$. Now $P_l \in \mathcal{S}_j$ implies that P_j had ensured that $A_l \subseteq \mathcal{A}_j$. This implies that $P_k \in M$ belongs to $\mathcal{A}_j$ before party P_j A-casted "Reconstruct Enabled". Since H_j is the instance of $\mathcal{A}_j$ at the time when P_j A-casts "Reconstruct Enabled", it is obvious that $P_k \in M$ belongs to H_j also. Using similar argument, it can be shown that *every* $P_k \in M$ also belong to H_j, thus proving the second part of the lemma.

Now we prove the third part of the lemma i.e $|M| \geq \frac{n}{3}$. A counting argument is used for this purpose. Let $m = |\mathcal{S}_i|$ at the time P_i A-casted "Reconstruct Enabled". So we have $m \geq n - t$. Now consider an $n \times n$ table Λ_i (relative to party P_i), whose l^{th} row and k^{th} column contains 1 for $k, l \in \{1, \ldots, n\}$ iff the following hold:

1. P_i has received "P_l Accepts A_l" from A-cast of P_l and included P_l in $\mathcal{S}_i$ before A-casting "Reconstruct Enabled" AND

2. $P_k \in A_l$

The remaining entries (if any) of Λ_i are left blank. Then M is the set of parties P_k such that k^{th} column in Λ_i contains 1 at least at $t+1$ positions. Notice that each row of Λ_i contains 1 at $n-t$ positions. Thus Λ_i contains 1 at $m(n-t)$ positions.

Let q denote the minimum number of columns in Λ_i that contain 1 at least at $t+1$ positions. We will show that $q \geq \frac{n}{3}$. The worst distribution of 1 entries in Λ_i is letting q columns to contain all 1 entries and letting each of the remaining $n-q$ columns to contain 1 at t locations. This distribution requires Λ_i to contain 1 at no more than $qm + (n-q)t$ positions. But we have already shown that Λ_i contains 1 at $m(n-t)$ positions. So we have

$$qm + (n-q)t \geq m(n-t).$$

This gives $q \geq \frac{m(n-t)-nt}{m-t}$. Since $m \geq n-t$ and $n \geq 3t+1$, we have

$$\begin{aligned} q &\geq \frac{m(n-t)-nt}{m-t} \geq \frac{(n-t)^2 - nt}{n-2t} \\ &\geq \frac{(n-2t)^2 + nt - 3t^2}{n-2t} \geq n - 2t + \frac{nt - 3t^2}{n-2t} \\ &\geq n - 2t + \frac{t}{n-2t} \geq \frac{n}{3} \end{aligned}$$

This shows that $|M| = q \geq \frac{n}{3}$ $\square$

Lemma 9.8 ([35]) *Let $\epsilon \leq 0.2$ and assume that all the honest parties have terminated protocol* Common-Coin*. Then for every value $\sigma \in \{0,1\}$, with probability at least $\frac{1}{4}$, all the honest parties output σ.*

Proof: By Lemma 9.6, for every party P_i that is included in $\mathcal{A}_j$ of some honest party P_j, there exists some fixed (yet unknown) value V_i that is distributed uniformly and independently over $[0,\ldots,u]$ and with probability $1 - \frac{\epsilon}{n}$ all honest parties will associate V_i with P_i. Consequently, as there are n^2 instances of WAVSS-Rec-Public, each with an error probability of $\epsilon' = \frac{\epsilon}{n^2}$, with probability at least $1 - n^2\epsilon' = (1-\epsilon)$, all honest parties will agree on the value associated with each one of the parties. Now we consider two cases:

- We now show that the probability of outputting $\sigma = 0$ by all honest parties is at least $\frac{1}{4}$. Let M be the set of parties discussed in Lemma 9.7. Clearly if $V_j = 0$ for some $P_j \in M$ and all honest parties associate V_j with P_j, then all the honest parties will output 0. The probability that for at least one party $P_j \in M$, $V_j = 0$ is $1 - (1-\frac{1}{u})^{|M|}$. Now recall that we assumed $u = \lceil 0.87n \rceil$. Also $|M| \geq \frac{n}{3}$ by Lemma 9.7. Therefore for all $n > 4$, we have $1 - (1-\frac{1}{u})^{|M|} \geq 0.316$. So, *Prob*(all honest parties output 0) $\geq 0.316 \times (1-\epsilon) \geq 0.25 = \frac{1}{4}$.
- We now show that the probability of outputting $\sigma = 1$ by all honest parties is at least $\frac{1}{4}$. It is obvious that if *no* party P_j has $V_j = 0$ and all honest parties associate V_j with P_j, then all honest parties will output 1. The probability of the first event is at least $(1-\frac{1}{u})^n \geq e^{-1.15}$. Thus *Prob*(all honest parties output 1) $\geq e^{-1.15} \times (1-\epsilon) \geq 0.25 = \frac{1}{4}$.

Hence the lemma. $\square$

Theorem 9.9 ([35]) *Protocol* Common-Coin *is a $(1-\epsilon)$-terminating, t-resilient common coin protocol for $n = 3t+1$ parties for every $0 < \epsilon \leq 0.2$.*

Proof: The **Termination** property (of Definition 9.4) follows from Lemma 9.5. The **Correctness** property (of Definition 9.4) follows from Lemma 9.6, Lemma 9.7 and Lemma 9.8. $\square$

Due to the use of efficient AVSS scheme in the place of relatively inefficient AVSS protocol of [35], protocol Common-Coin provides better communication complexity than the common coin protocol presented in [35].

Theorem 9.10 *Protocol* Common-Coin *privately communicates $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits and* A-casts *$\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits.*

Proof: Easy. Follows from Theorem 8.25 and the fact that Common-Coin executes at most n^2 instances of WAVSS-Share and WAVSS-Rec-Public, each with an error probability of $\frac{\epsilon}{n^2}$. $\square$

9.3.2 Existing Voting Protocol

The Voting protocol is another requirement for the construction of ABA protocol. In a Voting protocol, every party has a single bit as input. Roughly, Voting protocol tries to find out whether there is a detectable majority for some value among the inputs of the parties. Here we recall the Voting protocol called Vote from [35].

The Intuition: Each party's output in Vote protocol can take *five* different forms:

1. For $\sigma \in \{0, 1\}$, the output $(\sigma, 2)$ stands for 'overwhelming majority for σ';
2. For $\sigma \in \{0, 1\}$, the output $(\sigma, 1)$ stands for 'distinct majority for σ';
3. Output $(\Lambda, 0)$ stands for 'non-distinct majority'.

We will show that:

1. If all the honest parties have the same input σ, then all honest parties will output $(\sigma, 2)$;
2. If some honest party outputs $(\sigma, 2)$, then every other honest party will output either $(\sigma, 2)$ or $(\sigma, 1)$;
3. If some honest party outputs $(\sigma, 1)$ and no honest party outputs $(\sigma, 2)$ then each honest party outputs either $(\sigma, 1)$ or $(\Lambda, 0)$.

The Vote protocol consists of three stages, having similar structure. In the first stage, each party A-casts his input value, waits to receive $n - t$ A-casts of other parties, and sets his `vote` to the majority value among these inputs. In the second phase, each party A-casts his `vote` (along with the identities of the $n-t$ parties whose A-casted inputs were used to compute `vote`), waits to receive $n - t$ A-casts of other `vote`s that are consistent with the A-casted inputs of the first phase, and sets his `re-vote` to the majority value among these `vote`s. In the third phase, each party A-casts his `re-vote` along with the identities of the $n - t$ parties whose A-casted `vote`s were used to compute the `re-vote`, and waits to complete $n - t$ A-casts of other `re-votes` that are consistent with the consistent `vote`s of second phase. Now if all the consistent `vote`s received by a party agree on a value, σ, then the party outputs $(\sigma, 2)$. Otherwise, if all the consistent `re-vote`s received by the party agree on a value, σ, then the party outputs $(\sigma, 1)$. Otherwise, the party outputs $(\Lambda, 0)$. Protocol Vote is presented formally in Fig. 9.2. In the protocol, we assume party P_i has input bit x_i.

We now recall the proofs for the properties of protocol Vote from [35].

Lemma 9.11 ([35]) *All the honest parties terminate protocol Vote in constant time.*

PROOF: Every honest party P_i will eventually receive $(\texttt{input}, P_j, x_j)$ from the A-cast of every honest P_j. Thus every honest P_i will eventually have $|\mathcal{A}_i| = n-t$ and will A-cast $(\texttt{vote}, P_i, A_i, a_i)$. Now every honest party P_i will eventually receive $(\texttt{vote}, P_j, A_j, a_j)$ from the A-cast of every honest P_j. Thus every honest P_i will eventually have $|\mathcal{B}_i| = n - t$ and will A-cast $(\texttt{re-vote}, P_i, B_i, b_i)$. Now every honest party P_i will eventually receive
$(\texttt{re-vote}, P_j, B_j, b_j)$ from the A-cast of every honest P_j. Thus every honest P_i will eventually have $|C_i| = n - t$. Consequently, every honest P_i will terminate the protocol in constant time. □

Lemma 9.12 ([35]) *If all the honest parties have the same input σ, then all the honest parties will eventually output $(\sigma, 2)$ in protocol Vote.*

Figure 9.2: Existing Vote Protocol

Protocol Vote($\mathcal{P}$)

Code for P_i: — Every party executes this code

1. A-cast $(\texttt{input}, P_i, x_i)$.
2. Create a dynamic set $\mathcal{A}_i$. Add (P_j, x_j) to $\mathcal{A}_i$ if $(\texttt{input}, P_j, x_j)$ is received from the A-cast of P_j.
3. Wait until $|\mathcal{A}_i| = n - t$. Assign $A_i = \mathcal{A}_i$. Set a_i to the majority bit among $\{x_j \mid (P_j, x_j) \in A_i\}$ and A-cast $(\texttt{vote}, P_i, A_i, a_i)$.
4. Create a dynamic set $\mathcal{B}_i$. Add (P_j, A_j, a_j) to $\mathcal{B}_i$ if $(\texttt{vote}, P_j, A_j, a_j)$ is received from the A-cast of P_j, $A_j \subseteq \mathcal{A}_i$, and a_j is the majority bit of A_j.
5. Wait until $|\mathcal{B}_i| = n - t$. Assign $B_i = \mathcal{B}_i$. Set b_i to the majority bit among $\{a_j \mid (P_j, A_j, a_j) \in B_i\}$ and A-cast $(\texttt{re-vote}, P_i, B_i, b_i)$.
6. Create a set C_i. Add (P_j, B_j, b_j) to C_i if $(\texttt{re-vote}, P_j, B_j, b_j)$ is received from the A-cast of P_j, $B_j \subseteq \mathcal{B}_i$, and b_j is the majority bit of B_j.
7. Wait until $|C_i| \geq n - t$. If all the parties $P_j \in B_i$ had the same vote $a_j = \sigma$, then output $(\sigma, 2)$ and terminate. Otherwise, if all the parties $P_j \in C_i$ have the same `Re-vote` $b_j = \sigma$, then output $(\sigma, 1)$ and terminate. Otherwise, output $(\Lambda, 0)$ and terminate.

PROOF: Consider an honest party P_i. If all the honest parties have same input σ, then at most t (corrupted) parties may A-cast $\overline{\sigma}$ as their input. Therefore, it is easy to see every P_k (irrespective of whether honest or corrupted), who is included in $\mathcal{B}_i$ must have A-casted his `vote` $b_k = \sigma$. Hence honest P_i will output $(\sigma, 2)$. □

Lemma 9.13 ([35]) *If some honest party outputs $(\sigma, 2)$, then every other honest party will eventually output either $(\sigma, 2)$ or $(\sigma, 1)$ in protocol Vote.*

PROOF: Let an *honest* party P_i outputs $(\sigma, 2)$. This implies that all the parties $P_j \in B_i$ had A-casted the same `vote` $a_j = \sigma$. As the size of B_i is $n - t = 2t + 1$, it implies that for every other *honest* P_j, it holds that $|B_i \cap B_j| \geq t + 1$. This means that every other *honest* P_j is bound to A-cast re-vote b_i as σ. Hence every other honest party will eventually output either $(\sigma, 2)$ or $(\sigma, 1)$. □

Lemma 9.14 ([35]) *If some honest party outputs $(\sigma, 1)$ and no honest party outputs $(\sigma, 2)$ then every other honest party will eventually output either $(\sigma, 1)$ or $(\Lambda, 0)$ in protocol Vote.*

PROOF: Assume that some *honest* party P_i outputs $(\sigma, 1)$. This implies that all the parties $P_j \in C_i$ had A-casted the same `re-vote` $b_j = \sigma$. Since $|C_i| \geq n - t$, in the worst case there are *at most* t parties (outside C_i) who may A-cast `re-vote` $\overline{\sigma}$. Thus it is clear that no honest party will output $(\overline{\sigma}, 1)$. Now since the honest parties in C_i had `re-vote` as σ, there must be at least $t + 1$ parties who have

A-casted their `vote` as σ. Thus no honest party can output $(\overline{\sigma}, 2)$ for which at least $n - t = 2t + 1$ parties are required to A-cast their `vote` as $\overline{\sigma}$. So we have proved that no honest party will output from $\{(\overline{\sigma}, 2), (\overline{\sigma}, 1)\}$. Therefore the honest parties will output either $(\sigma, 1)$ or $(\Lambda, 0)$. □

Theorem 9.15 *Protocol Vote involves A-cast of $\mathcal{O}(n^2 \log n)$ bits.*

PROOF: In protocol Vote, each party P_i A-casts A_i and B_i, each containing the identity of $n - t = 2t + 1$ parties. Since the identity of each party can be represented by $\log n$ bits, protocol Vote involves A-cast of $\mathcal{O}(n^2 \log n)$ bits. □

9.3.3 The ABA Protocol for Single Bit

Once we have an efficient Common-Coin protocol and Vote protocol, we can design an efficient ABA protocol using the approach of [35]. The ABA protocol proceeds in iterations where in each iteration every party computes a 'modified input' value. In the first iteration the 'modified input' of party P_i is nothing but the private input bit x_i. In each iteration, every party executes two protocols *sequentially*: Vote and Common-Coin. That is protocol Common-Coin is executed only after the termination of Vote. If a party outputs $\{(\sigma, 2), (\sigma, 1)\}$ in Vote protocol, then he sets his 'modified input' for next iteration to σ, irrespective of the value which is going to be output in Common-Coin. Otherwise, he sets his 'modified input' for next iteration to be the output of Common-Coin protocol which is invoked by all the honest parties in each iteration irrespective of whether the output of Common-Coin is used or not. Once a party outputs $(\sigma, 2)$, he A-casts σ and once he receives $t + 1$ A-cast for σ, he terminates the ABA protocol with σ as final output. The protocol is formally presented in Fig. 9.3.

Our protocol has ϵ error in **Termination**. To bound the error probability by ϵ, the computation of ABA is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq 4n^6 2^{-\kappa}$. This is derived from the fact that in ABA, Common-Coin is invoked with $\frac{\epsilon}{4}$ error probability and as mentioned in Section 9.3.1, $\epsilon \geq n^6 2^{-\kappa}$ should hold to bound error probability of Common-Coin by ϵ. So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

We now state the following lemmas which are more or less identical to the Lemmas 5.36-5.39 presented in [35]. For the sake of completeness, we recall all of them with proofs.

Lemma 9.16 ([35]) *In protocol ABA if all honest parties have input σ, then all honest parties terminate and output σ.*

PROOF: If all honest parties have input σ, then by Lemma 9.12 every honest party will output $(y_1, m_1) = (\sigma, 2)$ upon termination of Vote and consequently A-cast (Terminatewith σ) in the first iteration. Therefore every honest party will eventually receive $n - t$ A-cast of (Terminate with σ). Hence every honest party will terminate ABA with σ as output. □

Lemma 9.17 ([35]) *In protocol ABA, if an honest party terminates with output σ, then all honest parties will eventually terminate with output σ.*

PROOF: To prove the lemma, we show that if an honest party A-casts (Terminate with σ), then eventually every other honest party will A-cast (Terminate with σ). Let k be the first iteration when an honest party P_i A-casts (Terminate with σ).

Figure 9.3: Efficient ABA Protocol for Single Bit.

Protocol ABA($\mathcal{P}, \epsilon$)

Code for P_i: Every party executes this code

1. Set r = 0. and $v_1 = x_i$.
2. Repeat until terminating.
   (a) Set $r = r + 1$. Invoke Vote($\mathcal{P}$) with v_r as input. Wait to terminate Vote and assign the output of Vote to (y_r, m_r).
   (b) Invoke Common-Coin($\mathcal{P}, \frac{\epsilon}{4}$) and wait until its termination. Let c_r be the output of Common-Coin.
   (c) i. If $m_r = 2$, set $v_{r+1} = y_r$ and A-cast (Terminate with v_{r+1}). Participate in *only one more* instance of Vote and *only one more* instance of Common-Coin protocol. /* The purpose of this restriction is to prevent the parties from participating in an unbounded number of iterations before enough (Terminate with σ) A-casts are completed.*/
   ii. If $m_r = 1$, set $v_{r+1} = y_r$.
   iii. Otherwise, set $v_{r+1} = c_r$.
   (d) Upon receiving $t+1$ (Terminate with σ) A-cast for some value σ, output σ and terminate ABA.

Then we prove that every other honest party will A-cast (Terminate with σ) either in k^{th} iteration or in $(k+1)^{th}$ iteration. Since honest P_i has A-casted (Terminate with σ), it implies that $y_k = \sigma$ and $m_k = 2$ and P_i has outputted $(\sigma, 2)$ in the Vote protocol invoked in k^{th} iteration. By Lemma 9.13, every other honest party P_j will output either $(\sigma, 2)$ or $(\sigma, 1)$ in the Vote protocol invoked in k^{th} iteration. In case P_j outputs $(\sigma, 2)$, the it will A-cast (Terminate with σ) in k^{th} iteration itself. Furthermore every honest P_j will execute Vote with input $v_{k+1} = \sigma$ in the $(k+1)^{th}$ iteration. So clearly, in $(k+1)^{th}$ iteration every honest party will have same input σ. Therefore by Lemma 9.12, every honest party will output $(\sigma, 2)$ in Vote protocol invoked in $(k+1)^{th}$ iteration. Hence all the honest parties will A-cast (Terminate with σ) either in iteration k or iteration $k+1$.

As all the honest parties will eventually A-cast (Terminate with σ), every honest party will receive $n - t$ A-casts of (Terminate with σ) and at most t A-casts of (Terminate with $\overline{\sigma}$). Therefore every honest party will eventually output σ. □

Lemma 9.18 ([35]) *If all honest parties have initiated and completed some iteration k, then with probability at least $\frac{1}{4}$ all honest parties have same value for 'modified input' v_{k+1}.*

Proof: We have two cases here:

1. If all honest parties execute step 4(c) in iteration k, then they have set their v_{k+1} as the output of protocol Common-Coin. So by the property of Common-Coin, all the honest party have same v_{k+1} with probability at least $\frac{1}{4}$.

2. If some honest party has set $v_{k+1} = \sigma$ for some $\sigma \in \{0, 1\}$, either in step 4(a) or step 4(b) of iteration k, then by Lemma 9.14 no honest party will set $v_{k+1} = \overline{\sigma}$ in step 4(a) or step 4(b). Moreover, all the honest honest parties will output σ from Common-Coin with probability at least $\frac{1}{4}$. Now the parties starts executing Common-Coin, only after the termination of Vote. Hence the outcome of Vote is *fixed* before Common-Coin is invoked. Thus corrupted parties can not decide the output of Vote to prevent agreement. Hence with probability at least $\frac{1}{4}$, all the honest parties will set $v_{k+1} = \sigma$. □

Let C_k be the event that each honest party completes all the iterations he initiated up to (and including) the k^{th} iteration (that is, for each iteration $1 \leq l \leq k$ and for each party P, if P initiated iteration l then he computes v_{l+1}). Let C denote the event that C_k occurs for all k.

Lemma 9.19 ([35]) *Conditioned on the event C, all honest parties terminate protocol ABA in constant expected time.*

PROOF: We first show that all the honest parties terminate protocol ABA within constant time after the *first* instance of A-cast of (Terminate with σ) is initiated by some *honest* party. Let the *first* instance of A-cast of (Terminate with σ) is initiated by some *honest* party in iteration k. Then all the parties will participate in Vote and Common-Coin protocols of all iterations up to iteration $k + 1$. Both the executions can be completed in constant time. Moreover, by the proof of Lemma 9.17 every honest party will A-cast (Terminate with σ) by the end of iteration $k + 1$. These A-casts can be completed in constant time. Since an honest party terminates ABA after completing $t + 1$ such A-casts, all the honest parties will terminate ABA within constant time after the *first* instance of A-cast of (Terminate with σ) is initiated by some *honest* party.

Now let the random variable τ be the count of number of iterations until the *first* instance of A-cast of (Terminate with σ) is initiated by some *honest* party. Obviously if no honest party ever A-casts (Terminate with σ) then $\tau = \infty$. Now conditioned on event C, all the honest parties terminate each iteration in constant time. So it is left to show that $E(\tau|C)$ is constant. We have

$$\begin{aligned} Prob(\tau > k|C_k) &\leq Prob(\tau \neq 1|C_k) \times \ldots \\ &\times Prob(\tau \neq k \cap \ldots \cap \tau \neq 1|C_k) \end{aligned}$$

From Lemma 9.18, it follows that each one of the k multiplicands of the right hand side of the above equation is at most $\frac{3}{4}$. Thus we have $Prob(\tau > k|C_k) \leq (\frac{3}{4})^k$. Now it follows by simple calculation that $E(\tau|C) \leq 16$. □

Lemma 9.20 ([35]) *$Prob(C) \geq (1 - \epsilon)$.*

PROOF: We have

$$\begin{aligned} Prob(\overline{C}) &\leq \sum_{k \geq 1} Prob(\tau > k \cap \overline{C_{k+1}}|C_k) \\ &\leq \sum_{k \geq 1} Prob(\tau > k|C_k) \cdot Prob(\overline{C_{k+1}}|C_k \cap \tau > k) \end{aligned}$$

From the proof of Lemma 9.18, we have $Prob(\tau > k|C_k) \leq (\frac{3}{4})^k$. We will now bound $Prob(\overline{C_{k+1}}|C_k \cap \tau \geq k)$. If all the honest parties execute the k^{th} iteration and complete the k^{th} invocation of Common-Coin, then all the honest parties complete k^{th} iteration. Protocol Common-Coin is invoked with termination parameter $\frac{\epsilon}{4}$. Thus with probability $1 - \frac{\epsilon}{4}$, all the honest parties complete the k^{th} invocation of Common-Coin. Therefore, for each k, $Prob(\overline{C_{k+1}}|C_k \cap \tau \geq k) \leq \frac{\epsilon}{4}$. So we get

$$Prob(\overline{C}) \leq \sum_{k \geq 1} \frac{\epsilon}{4} (\frac{3}{4})^k = \epsilon$$

The above equation implies that $Prob(C) \geq (1 - \epsilon)$. □
Summing up, we have the following theorem.

Theorem 9.21 (ABA for Single Bit) *Let $n = 3t+1$. Then for every $0 < \epsilon \leq 0.2$, protocol ABA is a $(\epsilon, 0)$-ABA protocol for n parties. Given the parties terminate, they do so in constant expected time. The protocol privately communicates $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits and A-casts $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits.*

Proof: The properties of ABA follows from Lemma 9.16, Lemma 9.17, Lemma 9.18 and Lemma 9.19. Let $\mathcal{C}$ be the expected number of time Common-Coin and Vote protocol are executed in ABA protocol. Then from Theorem 9.10 protocol ABA privately communicates $\mathcal{O}(\mathcal{C}n^6 \log \frac{1}{\epsilon})$ bits and A-casts $\mathcal{O}(\mathcal{C}n^6 \log \frac{1}{\epsilon})$ bits. As $\mathcal{C} = \mathcal{O}(1)$, the ABA protocol privately communicates $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits and A-casts $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits. □

9.4 Our Efficient ABA Protocol for Multiple Bits

Till now we have concentrated on the construction of efficient ABA protocol that allows the parties to agree on a *single* bit. We now present another efficient ABA protocol called ABA-MB [1], which achieves agreement on $n - 2t = t + 1$ bits *concurrently.* Notice that we could execute protocol ABA (presented in Section 9.3.3) $t + 1$ times *in parallel* to achieve agreement on $t + 1$ bits. From Theorem 9.21, this would require a private communication as well as A-cast of $\mathcal{O}(n^7 \log \frac{1}{\epsilon})$ bits. However surprisingly our protocol ABA-MB requires private communication and A-cast of $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits for the same task. Consequently, in protocol ABA-MB, the *amortized* cost to reach agreement on a *single* bit is $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits of private and A-cast communication.

In real-life applications typically ABA protocols are invoked on *long messages* rather than on single bit. Even in asynchronous multiparty computation (AMPC) [21, 35, 13], where typically lot of ABA invocations are required, many of the invocations can be parallelized and optimized to a single invocation with a long message. Hence ABA protocols with long message are very relevant to many real life situations. All existing protocols for ABA [136, 18, 29, 66, 67, 39, 35, 1, 127] are designed for single bit message. A naive approach to design ABA for $\ell > 1$ bit message is to parallelize ℓ invocations of existing ABA protocols dealing with single bit. This approach requires a communication complexity that is ℓ times the communication complexity of the existing protocols for single bit and hence is inefficient. In this chapter, we provide a far better way to design an ABA with multiple bits. For ℓ bits message with $\ell \geq t + 1$, we may break the message in to

[1] Here MB stands for multiple bits.

blocks of $t+1$ bits and invoke one instance of our ABA-MB for each one of the $t+1$ blocks.

To design our protocol ABA-MB, we consider our weak statistical AVSS scheme WAVSS-MS (consisting of sub-protocols (WAVSS-MS-Share,WAVSS-MS-Rec-Public)) to share $\ell \geq 1$ secrets *simultaneously* (presented in Sections 8.4.3 and 8.4.4 in Chapter 8). Then we incorporate protocol WAVSS-MS in the common coin protocol presented in Section 9.3.1 and show that it does not work. After that we present a new common coin protocol that can use WAVSS-MS a black box and show that the new common coin is much more efficient than the common coin that used WAVSS as a black box in Section 9.3.1. Finally, we construct our ABA protocol using the new common coin (with a slight change due to the use of new common coin.)

9.4.1 An Incorrect Common Coin Protocol

In Sections 8.4.3 and 8.4.4 in Chapter 8, we have presented an AVSS scheme called WAVSS-MS (consisting of sub-protocols WAVSS-MS-Share, WAVSS-MS-Rec-Public) that can share and reconstruct *multiple* secrets simultaneously and therefore it is much more communication efficient than multiple executions of AVSS scheme WAVSS for sharing and reconstructing *single* secret. In section 9.3.1, we had recalled Common-Coin protocol from [35] that uses our protocols WAVSS-Share and WAVSS-Rec-Public as black box. Specifically, each party in Common-Coin invokes n instances of protocol WAVSS-Share each sharing a single secret. Simple thinking would suggest that those n instances of protocol WAVSS-Share, each sharing a single secret could be replaced by more efficient single instance of WAVSS-MS-Share, sharing n secrets simultaneously. This would naturally lead to more efficient common coin protocol, which would further imply more efficient ABA protocol. In the following, we do the same in protocol Common-Coin-Wrong. But as the name suggests, we then show that this direct replacement of WAVSS-Share by WAVSS-MS-Share without further modification will lead to an incorrect common coin protocol (i.e Common-Coin-Wrong is not a correct common coin protocol). In what follows, we first describe Common-Coin-Wrong and then point out the exact property where Common-Coin-Wrong deviates from Common-Coin. This will imply that Common-Coin-Wrong is not a correct solution for a common coin protocol. Protocol Common-Coin-Wrong is given in Fig. 9.4.

We now show that protocol Common-Coin-Wrong does not satisfy Lemma 9.6 which will further imply that Common-Coin-Wrong is not a correct common coin protocol. Specifically though it is true that: once some honest party P_j receives "`Attach` T_i `to` P_i" from the A-cast of P_i and includes P_i in $\mathcal{A}_j$, a unique value V_i is fixed such that any honest party will `associate` V_i with P_i; but now it is no longer ensured that V_i is distributed uniformly over $[0, \ldots, u]$. That is the adversary $\mathcal{A}_t$ can decide V_i for up to $t-1$ honest parties and thus those V_i are no longer random and uniformly distributed over $[0, \ldots, u]$. Consequently, $\mathcal{A}_t$ can enforce *some honest parties* to *always output* 0, while other honest parties *may output* $\sigma \in \{0, 1\}$ with probability at least $\frac{1}{4}$. This will strictly violate the property of common coin that *every honest party* should output $\sigma \in \{0, 1\}$ with probability at least $\frac{1}{4}$.

Let P_i be an honest party. We now describe a specific behavior of $\mathcal{A}_t$ in Common-Coin-Wrong which would allow $\mathcal{A}_t$ to decide V_i to be 0 and thus make

Figure 9.4: An Incorrect Common Coin protocol obtained by replacing WAVSS-Share and WAVSS-Rec-Public by WAVSS-MS-Share and WAVSS-MS-Rec-Public respectively in Protocol Common-Coin

Protocol Common-Coin-Wrong($\mathcal{P}, \epsilon$)

Code for P_i: — Every party in $\mathcal{P}$ executes this code.

1. For $j = 1, \ldots, n$, choose a random value x_{ij} and execute WAVSS-MS-Share($P_i, \mathcal{P}, (x_{i1}, \ldots, x_{in}), \epsilon'$) where $\epsilon' = \frac{\epsilon}{n}$.

2. Participate in WAVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) for every $j \in \{1, \ldots, n\}$. We denote WAVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) by WAVSS-MS-Share$_j$.

3. Create a dynamic set $\mathcal{T}_i$. Add party P_j to $\mathcal{T}_i$ if WAVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) has been completed. Wait until $|\mathcal{T}_i| = t + 1$. Then assign $T_i = \mathcal{T}_i$ and A-cast "`Attach` T_i `to` P_i". We say that the secrets $\{x_{ji} | P_j \in T_i\}$ are the secrets `attached` to party P_i.

4. Create a dynamic set $\mathcal{A}_i$. Add party P_j to $\mathcal{A}_i$ if the following holds:

   (a) "`Attach` T_j `to` P_j" is received from the A-cast of P_j and

   (b) $T_j \subseteq \mathcal{T}_i$.

   Wait until $|\mathcal{A}_i| = n - t$. Then assign $A_i = \mathcal{A}_i$ and A-cast "P_i `Accepts` A_i".

5. Create a dynamic set $\mathcal{S}_i$. Add party P_j to $\mathcal{S}_i$ if the following holds:

   (a) "P_j `Accepts` A_j" is received from the A-cast of P_j and

   (b) $A_j \subseteq \mathcal{A}_i$.

   Wait until $|\mathcal{S}_i| = n - t$. Then A-cast "`Reconstruct Enabled`". Let H_i be the current content of $\mathcal{A}_i$.

6. Participate in WAVSS-MS-Rec-Public($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) for every $P_k \in T_j$ of every $P_j \in \mathcal{A}_i$ (Note that some parties may be included in $\mathcal{A}_i$ after the A-cast of "`Reconstruct Enabled`". The corresponding WAVSS-MS-Rec-Public are invoked immediately). We denote WAVSS-MS-Rec-Public($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) by WAVSS-MS-Rec-Public$_k$.

7. Let $u = \lceil 0.87n \rceil$. Every party $P_j \in \mathcal{A}_i$ is `associated` with a value, say V_j which is computed as follows: $V_j = (\sum_{P_k \in T_j} x_{kj})\ mod\ u$ where x_{kj} is reconstructed back after executing WAVSS-MS-Rec-Public($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) (if $NULL$ is reconstructed in instance WAVSS-MS-Rec-Public$_k$ then some predefined values $(x^\star_{k1}, \ldots, x^\star_{kn}) \in \mathbb{F}^n$ will be taken as the secrets; for details see Remark 8.28).

8. Wait until the values `associated` with all the parties in H_i are computed. Now if there exits a party $P_j \in H_i$ such that $V_j = 0$, then output 0. Otherwise output 1.

honest P_i to output 0 (this can be extended for $t-1$ honest P_is) whereas the remaining honest parties output $\sigma \in \{0,1\}$ with probability at least $\frac{1}{4}$. It is known that there are t parties under the control of $\mathcal{A}_t$. The specific behavior is given in Fig. 9.5.

The Reason for the Problem: The adversary behavior specified in Fig. 9.5 become possible due to the fact that a corrupted party is able to commit his secrets for party P_i even after knowing what other parties has committed for P_i. This was strictly controlled in Common-Coin, where a corrupted party did not have any information about the secrets committed by other parties for P_i, while committing his own secret for P_i. In Common-Coin, secrets associated with P_i (that is the secrets corresponding to T_i) were disclosed only after T_i was fixed. This was possible as every party $P_k \in T_i$ committed their secrets independently using different instance of WAVSS-Share. Thus as per requirement, corresponding WAVSS-Rec-Public was invoked to reconstruct the desired secret.

The above is not possible in Common-Coin-Wrong, because of simultaneous commitment and disclosure of n secrets in our WAVSS-MS-Share and WAVSS-MS-Rec-Public. So a party P_l containing P_k in T_l may A-cast "`Reconstruct Enabled`" early and starts executing P_k's instance of WAVSS-MS-Rec-Public. This process will disclose the desired secret x_{kl}; but at the same time it will disclose other undesired secrets assigned to other parties. Now later the adversary may always schedule messages such that P_i includes such P_k's in T_i and some other corrupted parties who have seen the secrets committed by P_k for P_i and then has committed his own secrets. This clearly shows that the adversary can completely control the final output of P_i by deciding the value to be associated with P_i.

The above problem can be eliminated if we can ensure that no corrupted party can ever commit any secret after a single honest party starts reconstructing secrets. This is what we have achieved in our new common coin protocol presented in the next section.

9.4.2 A New and Efficient Common Coin Protocol for Multiple Bits

In this section, we show how to twist protocol Common-Coin, so that it can handle the problem described in the previous section and can still use protocols WAVSS-MS-Share and WAVSS-MS-Rec-Public as black-boxes. Before that we first extend the basic definition of common coin for multiple bit binary output.

Definition 9.22 (Multi-Bit Common Coin) *Let π be an asynchronous protocol, where each party has local random input and ℓ bit output, where $\ell \geq 1$. We say that π is a $(1-\epsilon)$-terminating, t-resilient, multi-bit common coin protocol if the following requirements hold for every adversary $\mathcal{A}_t$:*

1. **Termination:** *With probability $(1-\epsilon)$, all honest parties terminate.*
2. **Correctness:** *For every $l = 1, \ldots, \ell$, all honest parties output σ_l with probability at least $\frac{1}{4}$ for every value of $\sigma_l \in \{0,1\}$.*

The Intuition: We now present a multi-bit common coin protocol, called Common-Coin-MB. Protocol Common-Coin-MB goes almost in the same line as Common-Coin-Wrong except that we add some more steps and modify some of the steps due to which the corrupted parties are forced to commit/share their secrets much before they can reconstruct and access anybody elses' secrets. Thus contrary to

Figure 9.5: Specific Adversary Behavior in Protocol Common-Coin-Wrong

Possible Behavior of $\mathcal{A}_t$ in Protocol Common-Coin-Wrong($\mathcal{P}, \epsilon$)

1. Except a single corrupted party P_j, $\mathcal{A}_t$ asks all the remaining $t-1$ corrupted parties to participate in Common-Coin-Wrong honestly. P_j is asked to honestly participate in the instances of WAVSS-MS-Share and WAVSS-MS-Rec-Public initiated by every other party acting as a dealer. But P_j is directed to *hold back* his invocation of WAVSS-MS-Share as a dealer.

2. $\mathcal{A}_t$ being the scheduler in the network, stops all the messages sent to P_i and sent by P_i, except the messages related to P_i's instance of WAVSS-MS-Share and WAVSS-MS-Rec-Public (this will not stop P_i to be part of anybody else's $\mathcal{T}_j$), until the following happens:

   (a) $n-t-1$ honest parties (except P_i) and $t-1$ corrupted parties (except P_j) carry out steps of Common-Coin-Wrong honestly, construct respective sets, A-cast "`Reconstruct Enabled`" and start invoking corresponding WAVSS-MS-Rec-Public$_k$ protocols.

   (b) This way the n secrets of each of $n-t-1$ honest parties (except P_i) and $t-1$ corrupted parties will be revealed. /* It is to be noted that the corrupted parties can successfully reconstruct secrets in WAVSS-MS-Rec-Public by behaving honestly even if the honest P_i does not participate in WAVSS-MS-Rec-Public.*/

   (c) Now $\mathcal{A}_t$ constructs a set T_i of size $t+1$ containing any t honest parties whose shared values $(x_{k1}, \ldots, x_{kn})$ are already disclosed to him plus corrupted party P_j.

   (d) Now $\mathcal{A}_t$ selects x_{ji} such that $V_i = (\sum_{P_k \in T_i} x_{kj}) \ \ mod \ \ u = 0$ and asks P_j to invoke WAVSS-MS-Share$_j$ with x_{ji} as the secret assigned to P_i.

3. $\mathcal{A}_t$ now schedules the messages to P_i such that P_i A-casts "`Attach` T_i `to` P_i" and eventually includes P_i in $\mathcal{A}_i$. So clearly H_i will contain P_i and hence P_i will output 0 since V_i is 0.

protocol Common-Coin-Wrong, the values associated with every party P_i are now indeed random and are uniformly distributed over $[0, \dots, u]$.

Precisely, we do the following in Common-Coin-MB. Each party acts as a dealer and shares n random secrets, using a single instance of WAVSS-MS-Share with allowed error probability of $\epsilon' = \frac{\epsilon}{n}$. The i^{th} secret shared by *each* party is associated with party P_i. Now a party P_i adds a party P_j to $\mathcal{T}_i$, only when at least $n-t$ parties have terminated P_j's instance of WAVSS-MS-Share. Recall that in Common-Coin-Wrong, a party P_i adds a party P_j to $\mathcal{T}_i$, when he himself has terminated P_j's instance of WAVSS-MS-Share. After that party P_i constructs $\mathcal{T}_i$, $\mathcal{A}_i$ and $\mathcal{S}_i$ and A-cast T_i, A_i and "`Reconstruct Enabled`" in the same way as performed in Common-Coin-Wrong, except a single difference that here P_i ensures T_i to contain $n-t$ parties (contrary to $t+1$ parties in Common-Coin-Wrong). The reason for enforcing $|T_i| = n - t$ is to obtain multiple bit output in protocol Common-Coin-MB and will be clear in the sequel. Now what follows is the most important step of Common-Coin-MB. Party P_i starts participating in WAVSS-MS-Rec-Public of the parties who are in his $\mathcal{T}_i$ only after receiving at least $n-t$ "`Reconstruct Enabled`" A-casts. Moreover party P_i halts execution of all the instances of WAVSS-MS-Share corresponding to the parties not in $\mathcal{T}_i$ currently and later resume them only when they are included in $\mathcal{T}_i$. This step along with the step for constructing $\mathcal{T}_i$ will ensure the desired property that in order to be part of any honest party's $\mathcal{T}_i$, a corrupted party must have to commit his secrets well before the first honest party receives $n-t$ "`Reconstruct Enabled`" A-casts and starts reconstructing secrets. This ensures that a corrupted party who is in $\mathcal{T}_i$ of any honest party had no knowledge what so ever about the secrets committed by other honest parties at the time he commits to his own secrets.

Let us see, how our protocol steps achieve the above task. Let P_i be the *first honest party* to receive $n-t$ "`Reconstruct Enabled`" A-casts and start invoking reconstruction process. Also let P_k be a *corrupted party* who belongs to $\mathcal{T}_j$ of some honest party P_j. This means that at least $t+1$ honest parties have already terminated WAVSS-MS-Share instance of P_k (this is because P_j has added P_k in $\mathcal{T}_j$ only after confirming that $n-t$ parties have terminated P_k's instance of WAVSS-MS-Share). This further means that there is at least one honest party, say P_α, who terminated P_k's instance of WAVSS-MS-Share before A-casting "`Reconstruct Enabled`" (because if it not the case, then the honest party P_α would have halted the execution of P_k's instance of WAVSS-MS-Share for ever and would never terminate it). This indicates that P_k is already committed to his secrets before the first honest party receives $n-t$ "`Reconstruct Enabled`" A-casts and starts the reconstruction. A more detailed proof is given in Lemma 9.24.

Another important feature of protocol Common-Coin-MB is that it is a multi-bit common coin protocol. This is attained by using the ability of Vandermonde matrix [141, 52] for extracting randomness. As a result, we could associate $n-2t$ values with each P_i, namely $V_{i1}, \dots, V_{i(n-2t)}$ in Common-Coin-MB, while a single value V_i was associated with P_i in Common-Coin. This leads every party to output $\ell = n - 2t$ bits in protocol Common-Coin-MB. We will show that the amortized communication cost of generating a single bit output in protocol Common-Coin-MB is far better than the communication cost of Common-Coin. As described in the subsequent sections, this is a definite move towards the improvement of the communication complexity of ABA protocol. We now briefly recall Vandermonde matrix and then present protocol Common-Coin-MB.

Vandermonde Matrix and Randomness Extraction [141, 52]: Let $\beta_1, \ldots, \beta_c$ be distinct and publicly known elements. We denote an $(r \times c)$ Vandermonde matrix by $V^{(r,c)}$, where for $i = 1, \ldots, c$, the i^{th} column of $V^{(r,c)}$ is $(\beta_i^0, \ldots, \beta_i^{r-1})^T$. The idea behind extracting randomness using $V^{(r,c)}$ is as follows: without loss of generality, assume that $r > c$. Moreover, let $(x_1, \ldots, x_r)$ be such that:

1. *Any* c elements of it are completely random and are unknown to adversary $\mathcal{A}_t$.

2. The remaining $r - c$ elements are completely independent of the c elements and also known to $\mathcal{A}_t$.

Now if we compute $(y_1, \ldots, y_c) = (x_1, \ldots, x_r)V$, then $(y_1, \ldots, y_c)$ is a random vector of length c unknown to $\mathcal{A}_t$, extracted from $(x_1, \ldots, x_r)$ [141, 52]. This principle is used in protocol Common-Coin-MB, which is given in Fig. 9.6. $\diamondsuit$

To bound the error probability by ϵ, the computation of Common-Coin-MB is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^6 2^{-\kappa}$. This is derived from the fact that in Common-Coin-MB, WAVSS-MS is invoked with $\frac{\epsilon}{n^2}$ error probability and as mentioned in Section 8.4.4, $\epsilon \geq n^4 2^{-\kappa}$ should hold to bound error probability of WAVSS-MS by ϵ. So here each element from the field is represented by $\kappa = \log|\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

Let E be an event, defined as follows: All invocations of WAVSS-MS have been terminated properly. That is, if an honest party has terminated WAVSS-MS-Share, then n values, say $(s'_1, \ldots, s'_n)$ are fixed. All honest parties will terminate the corresponding invocation of WAVSS-MS-Rec-Public with output $(s'_1, \ldots, s'_n)$. Moreover if dealer D is honest then $(s'_1, \ldots, s'_n)$ is D's shared secrets. It is easy to see that event E occurs with probability at least $1 - n\epsilon' = 1 - \epsilon$.

We now prove the properties of protocol Common-Coin-MB.

Lemma 9.23 *All honest parties terminate Protocol* Common-Coin-MB *in constant time.*

PROOF: We structure the proof in the following way. We first show that assuming every honest party has A-casted "`Reconstruct Enabled`", every honest party will terminate protocol Common-Coin-MB in constant time. Then we show that there exists at least one honest party who will A-cast "`Reconstruct Enabled`". Consequently, we prove that if one honest party A-casts "`Reconstruct Enabled`", then eventually every other honest party will do the same.

So let us first prove the first statement. Assuming every honest party has A-casted "`Reconstruct Enabled`", it will hold that eventually every honest party P_i will receive $n-t$ A-casts of "`Reconstruct Enabled`" from $n-t$ honest parties and will invoke WAVSS-MS-Rec-Public corresponding to every party in $\mathcal{T}_i$. Now it remains to show that WAVSS-MS-Rec-Public protocols invoked by every honest party P_i will be terminated eventually. It clear that a party P_k that is included in $\mathcal{T}_i$ of some honest party P_i will be eventually included in $\mathcal{T}_j$ of every other honest party P_j. Hence if P_i participates in WAVSS-MS-Rec-Public$_k$, then eventually every other honest party will do the same and thus WAVSS-MS-Rec-Public$_k$ will be completed by every body. Now every honest party will terminate protocol

Figure 9.6: Multi-Bit Common Coin Protocol using Protocol WAVSS-MS-Share and WAVSS-MS-Rec-Public as Black-Boxes

Protocol Common-Coin-MB($\mathcal{P}, \epsilon$)

Code for P_i: — All parties execute this code

1. For $j = 1, \ldots, n$, choose a random value x_{ij} and execute WAVSS-MS-Share($P_i, \mathcal{P}, (x_{i1}, \ldots, x_{in}), \epsilon'$) where $\epsilon' = \frac{\epsilon}{n}$.
2. Participate in WAVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) for every $j \in \{1, \ldots, n\}$. We denote WAVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn}), \epsilon'$) by WAVSS-MS-Share$_j$.
3. Upon terminating WAVSS-MS-Share$_j$, A-cast "P_i `terminated` P_j".
4. Create a dynamic set $\mathcal{T}_i$. Add party P_j to $\mathcal{T}_i$ if "P_k `terminated` P_j" is received from the A-cast of at least $n - t$ P_k's. Wait until $|\mathcal{T}_i| = n - t$. Then assign $T_i = \mathcal{T}_i$ and A-cast "`Attach` T_i `to` P_i". We say that the secrets $\{x_{ji} | P_j \in T_i\}$ are the secrets `attached` to party P_i.
5. Create a dynamic set $\mathcal{A}_i$. Add party P_j to $\mathcal{A}_i$ if
   (a) "`Attach` T_j `to` P_j" is received from the A-cast of P_j and
   (b) $T_j \subseteq \mathcal{T}_i$.

   Wait until $|\mathcal{A}_i| = n - t$. Then assign $A_i = \mathcal{A}_i$ and A-cast "P_i `Accepts` A_i".
6. Create a dynamic set $\mathcal{S}_i$. Add party P_j to $\mathcal{S}_i$ if
   (a) "P_j `Accepts` A_j" is received from the A-cast of P_j and
   (b) $A_j \subseteq \mathcal{A}_i$.

   Wait until $|\mathcal{S}_i| = n - t$. Then A-cast "`Reconstruct Enabled`". Let H_i be the current content of $\mathcal{A}_i$.

   Halt all the instances of WAVSS-MS-Share$_j$ for all P_j who are are not yet included in current $\mathcal{T}_i$. Later resume all such instances of WAVSS-MS-Share$_j$'s if P_j is included in $\mathcal{T}_i$.
7. Wait to receive "`Reconstruct Enabled`" from A-cast of at least $n-t$ parties. Participate in WAVSS-MS-Rec-Public($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) for every $P_k \in \mathcal{T}_i$. We denote WAVSS-MS-Rec-Public($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) by WAVSS-MS-Rec-Public$_k$. Notice that as on when new parties are added to $\mathcal{T}_i$, P_i participates in corresponding WAVSS-MS-Rec-Public.
8. Let $u = \lceil 0.87n \rceil$. Every party $P_j \in \mathcal{A}_i$ is `associated` with $n - 2t$ values, say $V_{j1}, \ldots, V_{j(n-2t)}$ in the following way. Let x_{kj} for every $P_k \in T_j$ has been reconstructed. Let X_j be the $n - t$ length vector consisting of $\{x_{kj} \mid P_k \in T_j\}$. Then set $(v_{j1}, \ldots, v_{j(n-2t)}) = X_j \cdot V^{(n-t,n-2t)}$, where $V^{(n-t,n-2t)}$ is an $(n-t) \times (n-2t)$ Vandermonde Matrix. Now $V_{jl} = v_{jl} \mod u$ for $l = 1, \ldots, n - 2t$ (if $NULL$ is reconstructed in instance WAVSS-MS-Rec-Public$_k$ then some predefined values $(x^{\star}_{k1}, \ldots, x^{\star}_{kn}) \in \mathbb{F}^n$ will be taken as the secrets; for details see Remark 8.28).
9. Wait until $n - 2t$ values `associated` with all the parties in H_i are computed. Now for every $l = 1, \ldots, n-2t$ if there exits a party $P_j \in H_i$ such that $V_{jl} = 0$, then set 0 as the l^{th} binary output; otherwise set 1 as the l^{th} binary output. Finally output the $n - 2t$ length binary vector.

Common-Coin-MB after executing the remaining steps of Common-Coin-MB such as computing $V_{i1}, \ldots, V_{i(n-2t)}$ etc. Given event E, all invocations of WAVSS-MS-Rec-Public terminate in constant time. The black box protocol for A-cast terminates in constant time. This proves the first statement.

We next show that there is at least one honest party who will A-cast "`Reconstruct Enabled`". So assume that P_i is the *first honest party* to A-cast "`Reconstruct Enabled`". We will first show that this event will always take place. First notice that till P_i A-cast "`Reconstruct Enabled`", no honest party would halt any instance of WAVSS-MS-Share. By the termination property of WAVSS-MS-Share, every honest party will eventually terminate the instance of WAVSS-MS-Share of every other honest party. Hence for every honest party P_j, every honest P_i will eventually receive A-cast of "P_k `terminated` P_j" from $n-t$ honest P_k's. Thus as there are at least $n-t$ honest parties, for every honest party P_i, $\mathcal{T}_i$ will eventually contain at least $n-t$ parties and hence P_i will eventually A-cast "`Attach` T_i `to` P_i". Furthermore eventually P_i will receive "`Attach` T_j `to` P_j" from every *honest* P_j. Now it is obvious that every party P_k included in $\mathcal{T}_j$ will be eventually included in $\mathcal{T}_i$ and thus $T_j \subseteq \mathcal{T}_i$ will hold good. Therefore, every honest P_j will be eventually included in $\mathcal{A}_i$. Thus for an honest P_i, $\mathcal{A}_i$ will eventually be of size $n-t$ and hence P_i will A-cast "P_i `Accepts` A_i". Now following the similar argument as above, we can show that for an honest P_i, $\mathcal{S}_i$ will eventually be of size $n-t$ and hence P_i will A-cast "`Reconstruct Enabled`". After this, P_i may stop executing at most t instances of WAVSS-MS-Share corresponding to t parties.

Now we show that every other honest party P_j will also A-cast "`Reconstruct Enabled`" eventually. It is easy to see that every party that is included in $\mathcal{T}_i$ will also be included in $\mathcal{T}_j$ eventually. Now as P_i has already ensured that $\mathcal{T}_i$ contains at least $n-t$ parties, the same will hold good for P_j and P_j will eventually A-cast "`Attach` T_j `to` P_j". Furthermore, as P_i has already received "`Attach` T_j `to` P_j" from at least $n-t$ parties and checked that $T_j \subseteq \mathcal{T}_i$, eventually the same will hold for P_j and he will A-cast "P_j `Accepts` A_j". Following similar argument as above, P_j will A-cast "`Reconstruct Enabled`".

Given event E, all invocations of WAVSS-MS-Share terminate in constant time. The black box protocol for A-cast terminates in constant time. Thus every honest party will A-cast "`Reconstruct Enabled`" in constant time. Hence protocol Common-Coin-MB terminates in constant time. □

We now prove the following important lemma, which is at the heart of protocol Common-Coin-MB. The lemma shows that the specific adversary behavior as specified in Fig. 9.5 is not applicable in protocol Common-Coin-MB.

Lemma 9.24 *Let a corrupted party P_k is included in $\mathcal{T}_j$ of an honest P_j in protocol Common-Coin-MB. Then the values shared by P_k in WAVSS-MS-Share$_k$ are completely independent of the values shared by the honest parties.*

PROOF: Let P_i be the *first honest* party to receive A-cast of "`Reconstruct Enabled`" from at least $n-t$ parties and start participating in WAVSS-MS-Rec-Public corresponding to each party in $\mathcal{T}_i$. To prove the lemma, we first assert that a *corrupted* party P_k will never be included in $\mathcal{T}_j$ of *any* honest P_j if P_k invokes his WAVSS-MS-Share *only after* P_i starts participating in WAVSS-MS-Rec-Public corresponding to each party in $\mathcal{T}_i$. We prove this by contradiction. Let P_i has received "`Reconstruct Enabled`" from the set of parties $\mathcal{B}_1$ with $|\mathcal{B}_1| \geq n-t$. Moreover, assume P_k invokes his WAVSS-MS-Share only after P_i received "`Reconstruct Enabled`" from the parties in $\mathcal{B}_1$ and starts participating in WAVSS-MS-Rec-Public corresponding to each party in $\mathcal{T}_i$. Furthermore, assume that P_k is still in $\mathcal{T}_j$ of an honest P_j. Now $P_k \in \mathcal{T}_j$ implies that P_j must have received "P_m `terminated` P_k" from A-cast of at least $n-t$ P_m's, say $\mathcal{B}_2$. Now $|\mathcal{B}_1 \cap \mathcal{B}_2| \geq n-2t$ and thus the intersection set contains at least one honest party, say P_α, as $n = 3t+1$. This implies that honest $P_\alpha \in \mathcal{B}_1$ and must have terminated WAVSS-MS-Share$_k$ before A-casting "`Reconstruct Enabled`". Otherwise

P_α would have halted the execution of $\mathsf{WAVSS\text{-}MS\text{-}Share}_k$ and would never A-cast "P_α `terminated` P_k" (see step 6 in the protocol). This further implies that P_k must have invoked $\mathsf{WAVSS\text{-}MS\text{-}Share}_k$ before P_i starts participating in WAVSS-MS-Rec-Public protocols. But this is a contradiction to our assumption.

Hence if the corrupted P_k is included in $\mathcal{T}_j$ of *any* honest P_j then he must have invoked $\mathsf{WAVSS\text{-}MS\text{-}Share}_k$ before any WAVSS-MS-Rec-Public has been invoked by any honest party. Thus P_k will have no knowledge of the secrets shared by honest parties when he chooses his own secrets for $\mathsf{WAVSS\text{-}MS\text{-}Share}_k$. Hence the lemma. □

Lemma 9.25 *In protocol Common-Coin-MB, once some honest party P_j receives "`Attach` T_i `to` P_i" from the A-cast of P_i and includes P_i in $\mathcal{A}_j$, $n-2t$ unique values $V_{i1}, \ldots, V_{i(n-2t)}$ are fixed such that*

1. *Every honest party will `associate` $V_{i1}, \ldots, V_{i(n-2t)}$ with P_i, except with probability ϵ.*

2. *Each of $V_{i1}, \ldots, V_{i(n-2t)}$ is distributed uniformly over $[0, \ldots, u]$ and independent of the values `associated` with the other parties.*

PROOF: Once some honest party P_j receives "`Attach` T_i `to` P_i" from the A-cast of P_i and includes P_i in $\mathcal{A}_j$, $n-2t$ unique values $V_{i1}, \ldots, V_{i(n-2t)}$ are fixed. Here $V_{i1}, \ldots, V_{i(n-2t)}$ are defined following the step 8 of the protocol. We now prove the first part of the lemma. According to the lemma condition, $P_i \in \mathcal{A}_j$. This implies that $T_i \subseteq \mathcal{T}_j$. So honest P_j will participate in $\mathsf{WAVSS\text{-}MS\text{-}Rec\text{-}Public}_k$ corresponding to each $P_k \in T_i$. Moreover, eventually $T_i \subseteq \mathcal{T}_k$ and $P_i \in \mathcal{A}_k$ will be true for *every other honest P_k*. So, every other honest party will participate in $\mathsf{WAVSS\text{-}MS\text{-}Rec\text{-}Public}_k$ corresponding to each $P_k \in T_i$. Now by the property of WAVSS-MS-Rec-Public, each honest party will reconstruct x_{ki} at the completion of $\mathsf{WAVSS\text{-}MS\text{-}Rec\text{-}Public}_k$, except with probability ϵ' (recall that each instance of WAVSS-MS-Share, WAVSS-MS-Rec-Public has an associated error probability of ϵ' in termination). Thus, with probability $1-(n-t)\epsilon' \approx 1-\epsilon$, every honest party will associate $V_{i1}, \ldots, V_{i(n-2t)}$ with P_i.

We now prove the second part of the lemma. By Lemma 9.24, when T_i is fixed, the values that are shared by corrupted parties in T_i are completely independent of the values shared by the honest parties in T_i. Now, each T_i contains at least $n-2t$ honest parties and every honest partys' shared secrets are uniformly distributed and mutually independent. Hence by the property of Vandermonde matrix the values $v_{i1}, \ldots, v_{i(n-2t)}$ are completely random and thus $V_{i1}, \ldots, V_{i(n-2t)}$ are uniformly and independently distributed over $[0, \ldots, u]$. □

Lemma 9.26 *In protocol Common-Coin-MB, once an honest party A-casts "`Reconstruct Enabled`", there exists a set M such that:*

1. *For every party $P_j \in M$, some honest party has received "`Attach` T_j `to` P_j" from the A-cast of P_j.*

2. *When any honest party P_j A-casts "`Reconstruct Enabled`", then it will hold that $M \subseteq H_j$.*

3. *$|M| \geq \frac{n}{3}$.*

PROOF: The proof directly follows from the proof of Lemma 9.7 □

Lemma 9.27 *Let $\epsilon \leq 0.2$ and assume that all honest parties have terminated protocol* Common-Coin-MB*. Then for every $l \in \{1, \ldots, n-2t\}$, all honest parties output σ_l with probability at least $\frac{1}{4}$ for every value of $\sigma_l \in \{0, 1\}$.*

PROOF: By Lemma 9.25, for every party P_i that is included in $\mathcal{A}_j$ of some honest party P_j, there exists some fixed (yet unknown) values $V_{i1}, \ldots, V_{i(n-2t)}$ that are distributed uniformly over $[0, \ldots, u]$ and with probability $(1-\epsilon)$ all honest parties will associate those $n-2t$ with P_i. Consequently, with the same probability, all the honest parties will agree on the value associated with each one of the parties (as there are n instances of WAVSS-Rec-Public, each with an error probability of $\epsilon' = \frac{\epsilon}{n}$). Now for every l^{th} bit, we may run the same argument as given in the proof of Lemma 9.8. □

Theorem 9.28 *Protocol* Common-Coin-MB *is a $(1-\epsilon)$-terminating, t-resilient multi-bit common coin protocol with $n - 2t = t + 1$ bits output for $n = 3t + 1$ parties for every $0 < \epsilon \leq 0.2$.*

PROOF: The **Termination** property (of Definition 9.22) follows from Lemma 9.23. The **Correctness** property (of Definition 9.22) follows from Lemma 9.24, Lemma 9.25, Lemma 9.26 and Lemma 9.27. □

Theorem 9.29 *Protocol* Common-Coin-MB *privately communicates $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits and* A-casts *$\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits.*

PROOF: Easy. This follows from Theorem 8.29 (that states the communication complexity of WAVSS-MS in Section 8.4.4 in Chapter 8) and the fact that Common-Coin-MB executes n instances of WAVSS-MS-Share and WAVSS-MS-Rec-Public with $\ell = n$ secrets and having an error probability of $\frac{\epsilon}{n}$. □

Above theorem clearly leads to the following corollary.

Corollary 9.29.1 *The amortized communication cost of generating a single bit output in protocol* Common-Coin-MB *is $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits of private communication and* A-cast*.*

The above corollary shows that the amortized communication complexity of generating single bit output in Common-Coin-MB is $\mathcal{O}(n^2)$ times better than the communication cost of Common-Coin. In the next section, we use Common-Coin-MB to design an ABA protocol where the parties starts with a initial input of $n - 2t = t + 1$ bits and reach agreement on $t + 1$ bits *concurrently.*

9.4.3 Final ABA Protocol for Achieving Agreement on $t+1$ Bits Concurrently

Using our multi-bit common coin protocol, we now construct an ABA protocol, which allows the parties to reach agreement on multiple bits. Specifically, we design protocol ABA-MB, which attains agreement on $n - 2t = t + 1$ bits concurrently. So initially every party has a private input of $n - 2t$ bits. Let the $n - 2t$ bit input of P_i be denoted by $x_{i1}, \ldots, x_{i(n-2t)}$.

The Intuition: The high level idea of the protocol ABA-MB is similar to protocol ABA (given in Section 9.3.3). The ABA protocol proceeds in iterations where in each iteration every party computes his 'modified input', consisting of $n-2t$ bits. In the first iteration the 'modified input' of party P_i is nothing but the private input bits of P_i. In *each* iteration, every party executes the following protocols *sequentially*:

1. $n-2t$ parallel instances of Vote protocol, one corresponding to each bit of the 'modified input';

2. A *single* instance of Common-Coin-MB.

Notice that the parties participate in the instance of Common-Coin-MB, only after terminating all the $n-2t$ instances of Vote protocol. Now corresponding to l^{th} bit of his 'modified input', every party does the following computation: If the party outputs $(\sigma_l, 2)$ or $(\sigma_l, 1)$ in the l^{th} instance of Vote protocol, then he sets the l^{th} bit of his 'modified input' for next iteration to σ_l, irrespective of the l^{th} bit which is going to be output in Common-Coin-MB. Otherwise, he sets the l^{th} bit of his 'modified input' for next iteration to be the l^{th} bit, which is the output of Common-Coin-MB protocol. In case the party outputs $(\sigma_l, 2)$, he A-casts (σ_l, l) and once he receives $t+1$ A-casts for (σ_l, l), he concludes that agreement is reached for the l^{th} bit and therefore sets σ_l as the l^{th} output bit and performs no further computation related to l^{th} bit except for participating in the Common-Coin-MB instance of subsequent iterations. Finally, if a party concludes that agreement is reached on all the $t+1$ bits, he terminates the protocol ABA-MB. So essentially, in protocol ABA-MB, the parties parallely perform *almost* similar computation as in ABA, corresponding to each of the $t+1$ bits. However, instead of executing $n-2t$ instances of Common-Coin protocol, the parties execute *only a single instance* of Common-Coin-MB, which leads to the reduction in the communication complexity of protocol ABA-MB. The protocol is formally given in Fig. 9.7.

Our protocol has ϵ error in **Termination**. To bound the error probability by ϵ, the computation of ABA-MB is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq 4n^6 2^{-\kappa}$. This is derived from the fact that in ABA-MB, Common-Coin-MB is invoked with $\frac{\epsilon}{4}$ error probability and as mentioned in subsection 9.4.2, $\epsilon \geq n^6 2^{-\kappa}$ should hold to bound error probability of Common-Coin-MB by ϵ. So here each element from the field is represented by $\kappa = \log|\mathbb{F}| = \mathcal{O}(\log\frac{1}{\epsilon})$ bits.

We now prove the properties of protocol ABA-MB.

Lemma 9.30 *In protocol ABA-MB, if all the honest parties have input $\sigma_1, \ldots, \sigma_{n-2t}$, then all the honest parties terminate and output $\sigma_1, \ldots, \sigma_{n-2t}$.*

PROOF: Directly follows from Lemma 9.16 and protocol steps. □

Lemma 9.31 *If some honest party terminates protocol ABA-MB with output $\sigma_1, \ldots, \sigma_{n-2t}$, then all honest parties will eventually terminate ABA-MB with output $\sigma_1, \ldots, \sigma_{n-2t}$.*

PROOF: To prove the lemma, it is enough to show that for every $l = 1, \ldots, n-2t$, if an honest party terminates ABA-MB with output σ_l, then all honest parties will eventually terminate ABA-MB with output σ_l. However, this follows from the proof of Lemma 9.17. □

Figure 9.7: ABA Protocol to Reach Agreement on $n - 2t = t + 1$ Bits

Protocol ABA-MB($\mathcal{P}, \epsilon$)

Code for P_i: — Every party executes this code

1. Set $r = 0$. For $l = 1, \ldots, n - 2t$, set $v_{1l} = x_{il}$.
2. Repeat until terminating.
   (a) Set $r = r + 1$. Participate in $n - 2t$ instances of Vote protocol, with v_{rl} as the input in the l^{th} instance of Vote protocol, for $l = 1, \ldots, n - 2t$. Set (y_{rl}, m_{rl}) as the output of the l^{th} instance of Vote protocol.
   (b) Wait to terminate all the $n - 2t$ instances of Vote protocol. Then invoke Common-Coin-MB($\mathcal{P}, \frac{\epsilon}{4}$) and wait until its termination. Let $c_{r1}, \ldots, c_{r(n-2t)}$ be the output of Common-Coin-MB.
   (c) For every $l \in \{1, \ldots, n - 2t\}$ such that agreement on l^{th} bit is not achieved, do the following in parallel:
      i. If $m_{rl} = 2$, then set $v_{(r+1)l} = y_{rl}$ and A-cast ("Terminate with $v_{(r+1)l}$", l). Participate in only one more instance of Vote corresponding to l^{th} bit with $v_{(r+1)l}$ as the input. Participate in only one more instance of Common-Coin-MB if ("Terminate with $v_{(r+1)l}$", l) is A-casted for *all* $l = 1, \ldots, n - 2t$.
      ii. If $m_{rl} = 1$, set $v_{(r+1)l} = y_{rl}$.
      iii. Otherwise, set $v_{(r+1)l} = c_{rl}$.
   (d) Upon receiving ("Terminate with σ_l", l) from the A-cast of at least $t+1$ parties, for some value σ_l, output σ_l as the l^{th} bit and terminate all the computation regarding l^{th} bit. In this case, we say that agreement on l^{th} bit is achieved.
   (e) Terminate ABA-MB when agreement is achieved on all l bits, for $l = 1, \ldots, n - 2t$.

Lemma 9.32 *If all honest parties have initiated and completed some iteration k, then with probability at least $\frac{1}{4}$, all honest parties will have same value for 'modified input' $v_{(k+1)l}$, for every $l = 1, \ldots, n - 2t$.*

PROOF: Follows from the proof of Lemma 9.18. □

We now recall event C_k and C from section 9.3.3. Let C_k be the event that each honest party completes all the iterations he initiated up to (and including) the k^{th} iteration (that is, for each iteration $1 \le r \le k$ and for each party P, if P initiated iteration r then he computes $v_{(r+1)l}$ for every l^{th} bit). Let C denote the event that C_k occurs for all k.

Lemma 9.33 *Conditioned on event C, all honest parties terminate protocol ABA-MB in constant expected time.*

PROOF: Let the *first* instance of A-cast of ("Terminate with σ_l", l) is initiated by some honest party in iteration τ_l. Following Lemma 9.17, every other honest party will A-cast ("Terminate with σ_l", l) in iteration $\tau_l + 1$. Now it is true that agreement on l^{th} bit will be achieved within constant time after $(\tau_l+1)^{th}$ iteration (this is because the A-casts can be completed in constant time). Let m be such that τ_m is the maximum among $\tau_1, \ldots, \tau_{n-2t}$. We first show that all honest parties will terminate protocol ABA-MB within constant time after some honest party initiates the first instance of A-cast ("Terminate with σ_m", m). Since the first instance of A-cast of ("Terminate with σ_m", m) is initiated by some honest party in iteration τ_m, all the parties will participate in Vote and Common-Coin-MB in iteration $\tau_m + 1$. Both the executions can be completed in constant time. Moreover, by Lemma 9.17 every honest party will A-cast ("Terminate with σ_m", m) by the end of iteration $\tau_m + 1$. The A-casts can be completed in constant time. Moreover, it is to be noted that for all other bits l, agreement will be reached either before reaching agreement on m^{th} bit or within constant time of reaching agreement on m^{th} bit. Hence all honest parties will terminate ABA-MB within constant time after the *first* instance of A-cast of ("Terminate with σ_m", m) is initiated by some honest party in iteration τ_m.

Now conditioned on event C, all honest parties terminate each iteration in constant time. So it is left to show that $E(\tau_m|C)$ is constant. We have

$$\begin{aligned} Prob(\tau_m > k|C_k) &\leq Prob(\tau_m \neq 1|C_k) \times \\ &\ldots \times Prob(\tau_m \neq k \cap \ldots \cap \tau_m \neq 1|C_k) \end{aligned}$$

From the Lemma 9.32, it follows that each one of the k multiplicands of the right hand side of the above equation is at most $\frac{3}{4}$. Thus we have $Prob(\tau_m > k|C_k) \leq (\frac{3}{4})^k$. Now simple calculation gives $E(\tau_m|C) \leq 16$. □

Lemma 9.34 $Prob(C) \geq (1-\epsilon)$.

PROOF: Follows from the proof of Lemma 9.20. □

Summing up, we have the following theorem.

Theorem 9.35 (ABA for $t+1$ Bits) *Let $n = 3t+1$. Then for every $0 < \epsilon \leq 0.2$, protocol ABA-MB is a t-resilient, $(\epsilon, 0)$-ABA protocol for n parties. Given the parties terminate, they do so in constant expected time. The protocol allows the parties to reach agreement on $t+1$ bits simultaneously and involves private communication and A-cast of $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits.*

Corollary 9.35.1 *Protocol ABA-MB requires an amortized communication complexity of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits (private communication plus A-cast) for reaching agreement on a single bit.*

9.5 Conclusion and Open Problems

We have presented a novel, constant expected time, optimally resilient, $(\epsilon, 0)$-ABA protocol whose communication complexity is significantly better than the so far best known existing ABA protocols of [39, 1] (though the ABA protocol of [1] has a strong property of being *almost surely terminating*) with optimal resilience. Here we summarize the key factors that have contributed to the gain in the communication complexity of our ABA protocol:

- A shorter route: *ICP* $\rightarrow$ *AWSS* $\rightarrow$ *AVSS* $\rightarrow$ *ABA*,
- Improving each of the building blocks by introducing new techniques and
- By exploiting the advantages of dealing with multiple secrets concurrently in each of these blocks.

A few interesting open problems that are left here are:

Open Problem 15 *How to further improve the communication complexity of ABA protocols with optimal resilience?*

Open Problem 16 *Can we design an almost surely terminating, optimally resilient, constant expected time ABA protocol?*

9.6 APPENDIX: Analysis of the Communication Complexity of the ABA Scheme of [39, 35]

The communication complexity analysis of the ABA protocol of [39, 35] was not reported anywhere so far. So we have carried out the same at this juncture. Since AVSS is the main building block of the protocol, we require to know the communication complexity of the AVSS of [39, 35]. The analysis of communication complexity of the AVSS of [39, 35] was performed in section 8.7 of Chapter 8. We recall the complexity figures here:

- AVSS-Share protocol of [39] requires a communication complexity of $\mathcal{O}(n^9(\log\frac{1}{\epsilon})^4)$ bits and A-cast of $\mathcal{O}(n^9(\log\frac{1}{\epsilon})^2\log(n))$ bits.
- AVSS-Rec protocol requires a communication complexity of $\mathcal{O}(n^6(\log\frac{1}{\epsilon})^3)$ bits and A-cast of $\mathcal{O}(n^6(\log\frac{1}{\epsilon})\log(n))$ bits.

Now in the *common coin* protocol, each party in $\mathcal{P}$ acts as a dealer and invokes n instances of AVSS-Share to share n secrets. So the communication complexity of the common protocol of [39] is $\mathcal{O}(n^{11}(\log\frac{1}{\epsilon})^4)$ bits of private communication and $\mathcal{O}(n^{11}(\log\frac{1}{\epsilon})^2\log(n))$ bits of A-cast. Now in the ABA protocol of [39], AVSS-Share protocol is called for $\mathcal{C}=\mathcal{O}(1)$ expected time. Hence the ABA protocol of [39] involves a private communication of $\mathcal{O}(n^{11}(\log\frac{1}{\epsilon})^4)$ bits and A-cast of $\mathcal{O}(n^{11}(\log\frac{1}{\epsilon})^2\log(n))$ bits.

Chapter 10

Efficient Statistical AMPC with Optimal Resilience

In this chapter, we design a statistical AMPC protocol with optimal resilience i.e $n = 3t + 1$. Our protocol privately communicates $\mathcal{O}(n^5(\log \frac{1}{\epsilon}))$ bits per multiplication gate, where ϵ is the error probability. There is only one optimally resilient statistical AMPC protocol in the literature reported in [21]. The protocol of [21] privately communicates $\Omega(n^{11}(\log \frac{1}{\epsilon})^4)$ bits and A-casts $\Omega(n^{11}(\log \frac{1}{\epsilon})^2 \log(n))$ bits per multiplication gate. Thus our AMPC protocol significantly improves the communication complexity of only known optimally resilient statistically secure AMPC protocol of [21].

As a key tool of our AMPC, we design a new primitive called Asynchronous Complete Secret Sharing (ACSS). ACSS uses the strong statistical AVSS presented in Chapter 8 as a vital building block. Our ACSS may be used in many other applications and thus is of independent interest.

10.1 Introduction

The MPC problem has been studied extensively over synchronous networks [3, 5, 6, 7, 20, 12, 14, 9, 36, 41, 48, 49, 52, 95, 93, 98, 100, 102, 11, 101, 103, 104, 120, 138, 126, 153]. However, MPC in asynchronous network has got comparatively less attention, due to its inherent hardness. Since asynchronous networks model real life networks like Internet more appropriately than synchronous networks, fundamental problems like MPC is worthy of deep investigation over asynchronous networks.

10.1.1 Network and Adversary Model

This is same as described in section 8.1.1. Recall that the set of parties is denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$ and t out of the n parties can be under the influence of a *computationally unbounded Byzantine (active) adversary*, denoted as $\mathcal{A}_t$. We emphasize that we use $n = 3t + 1$ in this chapter.

10.1.2 Definitions

Asynchronous MPC or AMPC: An AMPC protocol allows the parties in $\mathcal{P}$ to securely compute an agreed function f, even in the presence of $\mathcal{A}_t$. More specifically, assume that the agreed function f can be expressed as $f : \mathbb{F}^n \rightarrow \mathbb{F}^n$ and party P_i has input $x_i \in \mathbb{F}$. Then the following should hold:

1. **Correctness:** At the end of the protocol, each honest P_i gets $y_i \in \mathbb{F}$, where $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, irrespective of the behavior of $\mathcal{A}_t$.

2. **Secrecy:** Moreover, $\mathcal{A}_t$ should not get any information about the input and output of the honest parties, other than what can be inferred from the input and output of the corrupted parties.

3. **Termination:** Every honest party should eventually terminate the protocol.

In any general AMPC protocol, the function f is specified by an arithmetic circuit over $\mathbb{F}$, consisting of input, linear (e.g. addition), multiplication, random and output gates. We denote the number of gates of these types in the circuit by c_I, c_A, c_M, c_R and c_O respectively. *Among all the different type of gates, evaluation of a multiplication gate requires the maximum communication complexity. So the communication complexity of any general AMPC protocol is usually given in terms of the communication complexity per multiplication gate* [14, 13, 12, 52, 106].

Definition 10.1 *A statistically secure (statistical in short) AMPC protocol involves a negligible error probability of ϵ in* **correctness** *and/or* **termination**. *However, note that there is no compromise in* **secrecy** *property.*

Typically, VSS is used as a tool for generating t-$(1d)$-sharing (for the definition of t-$(1d)$-sharing see Definition 6.12) of secret. That is, at the end of sharing phase, *each* honest party holds his share of the secret such that shares of *all* honest parties constitute distinct points on a degree-t polynomial. Such VSS protocols are reported in [20, 109]. On the other hand, there are VSS schemes that do not generate t-$(1d)$-sharing of secret at the end of sharing phase. They only ensure that a unique secret is shared / committed (during sharing phase) which will be uniquely reconstructed during reconstruction phase. Such schemes are presented in [73, 39]. Even the weak statistical AVSS and strong statistical AVSS protocols presented in Chapter 8 do not generate t-$(1d)$-sharing of secret(s). So we call a VSS scheme as *Complete Secret Sharing* (CSS) scheme if it generates t-$(1d)$-sharing of secret(s). More formally, we have the following definition for Statistical Asynchronous Complete Secret Sharing (ACSS):

Definition 10.2 (Statistical ACSS) *Let (Sh, Rec) be a pair of protocols in which a dealer $D \in \mathcal{P}$ shares a secret s using Sh. We say that (Sh[1], Rec[2]) is a t-resilient statistical ACSS scheme if it satisfies* **termination, correctness** *and* **secrecy** *property of strong statistical AVSS (see Definition 8.2). In addition, ACSS achieves the following* **Completeness** *property at the end of Sh with probability at least $(1 - \epsilon)$:*

- **Completeness**: *If some honest party terminates Sh, then there exists a random degree-t polynomial $f(x)$ over $\mathbb{F}$, with $f(0) = s'$ such that each (honest) party $P_i \in \mathcal{P}$ will eventually hold his share $s_i = f(i)$ of secret s'. Moreover, if D is honest, then $s' = s$.*

[1] Sh is the protocol for sharing phase of ACSS scheme
[2] Rec is the protocol for reconstruction phase of ACSS scheme

The above definition of statistical ACSS can be extended for secret S containing multiple elements (say ℓ with $\ell > 1$) from $\mathbb{F}$.

Remark 10.3 (ACSS with Private Reconstruction) *The definition of ACSS as given above consider "public reconstruction", where all parties publicly reconstruct the secret in* Rec*. A common variant of the definition consider "private reconstruction", where only some specific party, say $P_\alpha \in \mathcal{P}$, is allowed to reconstruct the secret in* Rec*. As per our requirement in this chapter, we present our ACSS schemes with both private as well as public reconstruction.*

10.1.3 Relevant History of Statistical Asynchronous MPC

From [21], *statistically secure* AMPC is possible iff $n \geq 3t+1$. In this chapter, we concentrate on *statistical* AMPC with *optimal resilience*, i.e., with $n = 3t+1$. The communication complexity *per multiplication gate* of existing statistical AMPC protocols are given in Table 10.1.

Table 10.1: Existing Statistical AMPC Protocols.

Reference	Resilience	Communication Complexity in bits Per Multiplication Gate
[21]	$t < n/3$ (optimal)	Private– $\Omega(n^{11}(\log \frac{1}{\epsilon})^4)$; A-cast– $\Omega(n^{11}(\log \frac{1}{\epsilon})^2 \log(n))$
[135]	$t < n/4$ (non-optimal)	Private– $\mathcal{O}(n^4(\log \frac{1}{\epsilon}))$

From Table 10.1, we find that the only known statistical AMPC with *optimal resilience* (i.e., with $n = 3t+1$), involves very high communication complexity (the communication complexity analysis of the AMPC of [21] was not done earlier and for the sake of completeness, we carry out the same in section 10.4). Recently [51] presented an efficient MPC protocol over networks that have a synchronization point (the network is asynchronous before and after the synchronization point) and hence we do not compare it with our AMPC protocol, which is designed over completely asynchronous settings. Also we do not compare our protocol with the known cryptographically secure AMPC (where the adversary has bounded computing power) protocols presented in [105] and [106].

10.1.4 Contribution of This Chapter

We design an optimally resilient statistical AMPC protocol that privately communicates $\mathcal{O}(n^5(\log \frac{1}{\epsilon}))$ bits per multiplication gate. Thus our AMPC protocol significantly improves the communication complexity of only known optimally resilient statistically secure AMPC protocol of [21].

For designing our AMPC protocol, we need a tool to generate t-$(1d)$-sharing of secrets. For this we propose an ACSS scheme which in turn uses our strong statistical AVSS protocol presented in Chapter 8.

Our AMPC protocol has error probability of ϵ. To bound the error probability by ϵ, all our protocols work over a finite field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq 2n^7 2^{-\kappa} \max(4n, \kappa)$. Each field element can be represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits (this can be derived using $n = \mathcal{O}(\log \frac{1}{\epsilon})$).

In order to bound the error probability of our AMPC protocol by some specific value of ϵ, we find out the *minimum* value of κ that satisfies the relation between κ and ϵ. The value for κ will consequently determine the field $\mathbb{F}$ over which the protocol should work.

10.1.5 The Road-map

Section 10.2 presents our ACSS protocol in detail (for simplicity, subsection 10.2.2 presents ACSS sharing single secret and Section 10.2.3 extends the ACSS for multiple secrets). Section 10.3 talks about the primitives that are used for our AMPC. Some of the primitives are constructed using our ACSS protocol. Section 10.4 presents a discussion on the approaches used in the AMPC of [21] (the protocol with which we compare our protocol) and the AMPC presented in this chapter. Next section 10.5 describes another important protocol that is to be used in our AMPC and that uses ACSS as black box. Subsequently, sections 10.6, 10.7, 10.8 and 10.9 are dedicated for our AMPC protocol. Lastly, this chapter ends with concluding remark and a set of interesting open questions in section 10.10

10.2 Statistical ACSS

For the sake of simplicity, we first present our ACSS protocol sharing a single secret and then extend the protocol for multiple (i.e ℓ) secrets. We will show that dealing with multiple secrets concurrently in our ACSS protocol provides with better communication complexity than multiple executions of protocol dealing with single secret. Prior to our discussion, we present the existing tool that will be used in our ACSS protocols.

10.2.1 Tool Used for our Statistical ACSS

Apart from A-cast that was recalled in Chapter 7, we require the following tool for our ACSS.

Online Error Correction (OEC): Let s be a secret which is t-$(1d)$-shared among the parties in $\mathcal{P}$ by a degree-t polynomial $f(x)$. So $f(0) = s$. Let $P_\alpha \in \mathcal{P}$ be a specific party, who wants to reconstruct s. Towards this every party P_i sends his share s_i of s to P_α. The shares may reach P_α in any arbitrary order. Moreover, up to t of the shares may be incorrect or missing. In such a situation, by applying OEC on the received s_i's, party P_α can get the interpolation polynomial $f(x)$ and reconstruct the secret $s = f(0)$ in an online fashion. The OEC method uses the properties of Reed-Solomon error correcting codes [119] and enables P_α to recognize when the received shares define a unique degree-t interpolation polynomial.

Since OEC is a very well known asynchronous primitive, we avoid giving complete details here. The interested reader can refer [35] for complete details.

10.2.2 Statistical ACSS for Sharing a Single Secret

So we now present an ACSS scheme called ACSS, which consists of sub-protocols (ACSS-Share, ACSS-Rec-Private, ACSS-Rec-Public). Protocol ACSS-Share allows

D to generate t-$(1d)$-sharing of a secret $s \in \mathbb{F}$. Given t-$(1d)$-sharing of secret s, protocol ACSS-Rec-Private allows a specific party in $\mathcal{P}$, say P_α, to privately reconstruct s. On the other hand, ACSS-Rec-Public allows every party in $\mathcal{P}$ to reconstruct D's committed secret s.

Protocol ACSS uses the strong statistical AVSS protocol called SAVSS (consisting of sub-protocols (SAVSS-Share, SAVSS-Rec-Private)) for sharing single secret, presented in Sections 8.5.1 and 8.5.2 of Chapter 8. Notice that though protocol SAVSS (and also SAVSS-MS) is an AVSS scheme, it is not an ACSS scheme as it does not achieve **completeness** property. The reason is that only the honest parties in $VCORE$ receive their respective shares of the committed secret in protocol SAVSS-Share. But it may happen that potentially t honest parties are *not* present in $VCORE$. This may cause that all the honest parties do not hold their shares of committed secret at the end of SAVSS-Share.

The Intuition: The high level idea of ACSS-Share is similar to SAVSS-Share. But now in ACSS-Share, we use SAVSS-Share as a black-box, in place of AWSS-Share. It is this change which helps ACSS to achieve *completeness property.* We now show that SAVSS-Share which uses AWSS-Share as a black box may not output t-$(1d)$-sharing of D's committed secret. Subsequently, we also point out how ACSS-Share overcome this problem by using SAVSS-Share as a black-box.

So let us consider protocol SAVSS-Share when D is *corrupted* and also assume that D is committed to a unique secret and thus a unique bi-variate polynomial $\overline{F}(x, y)$ of degree-(t, t). But in spite of this, we could only ensure that every honest P_i who A-casts `Matched-Row` signal, holds the corresponding row polynomial $f_i(x) = \overline{F}(x, i)$ (recall that we referred $f_i(x)$ polynomials as row polynomials and $g_i(y) = \overline{F}(i, y)$ polynomials as column polynomials; see Section 8.5) and hence his share $f_i(0)$ of the secret $\overline{s} = \overline{F}(0, 0)$. However, it is possible that there are potential t honest P_i's who have not A-casted `Matched-Row` signal due to the reconstruction of $NULL$ from P_i-weak-private-reconstruction during **Verification of D's Commitment Phase**. Also a corrupted D may not even pass on $\overline{F}(x, i)$ or may pass some wrong polynomial other than $\overline{F}(x, i)$ to these P_i's. So in this case t potential honest parties may not hold shares of secret $\overline{s}$.

On the other hand, SAVSS-Share is used as a black-box in ACSS-Share. This overcomes the above problem because now D would commit each $f_i(x)$ using SAVSS-Share, instead of AWSS-Share. So once it is ensured that D is committed to a unique bi-variate polynomial $\overline{F}(x, y)$ of degree-(t, t), by the property of SAVSS-Rec-Private, each honest $P_i \in \mathcal{P}$ would successfully reconstruct $f_i(x) = \overline{F}(x, i)$ and hence his share $f_i(0)$ of the secret $\overline{s} = \overline{F}(0, 0)$. Protocol ACSS-Share is provided in Fig. 10.1. Protocol ACSS-Rec-Private and ACSS-Rec-Public uses OEC (Online Error Correction method) and are presented in Fig. 10.2.

To bound the error probability by ϵ, the computation of ACSS is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^5 \kappa 2^{-\kappa}$. This is derived from the fact that in ACSS, protocol SAVSS is invoked with $\frac{\epsilon}{n}$ error probability and as mentioned in subsection 8.5.2, $\epsilon \geq n^4 \kappa 2^{-\kappa}$ should hold to bound error probability of SAVSS by ϵ. So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

We now prove the properties of ACSS.

Lemma 10.4 *In protocol ACSS-Share:*

Figure 10.1: Protocol ACSS-Share for Sharing Secret s with $n = 3t + 1$

Protocol ACSS-Share($D, \mathcal{P}, s, \epsilon$)

i. Distribution by D: Code for D – Only D executes this code

1. Select a random degree-(t, t) bivariate polynomial $F(x, y)$ such that $F(0, 0) = s$.
2. For $i = 1, \ldots, n$, send $g_i(y) = F(i, y)$ to party P_i. We call $g_i(y)$ as i^{th} column polynomial.
3. For $i = 1, \ldots, n$, initiate SAVSS-Share($D, \mathcal{P}, f_i(x), \epsilon'$) (See Notation 8.41 for the syntax) for sharing $f_i(x)$, where $f_i(x) = F(x, i)$ and $\epsilon' = \frac{\epsilon}{n}$. We call $f_i(x)$ as i^{th} row polynomial. We refer SAVSS-Share($D, \mathcal{P}, f_i(x), \epsilon'$) by SAVSS-Share$_i$.

ii. Code for P_i: – Every party in $\mathcal{P}$, including D, executes this code

1. Wait to receive degree-t column polynomial $g_i(y)$ from D.
2. Participate in SAVSS-Share$_j$ for all $j = 1, \ldots, n$.
3. If $f_j(i)$ is received from D during SAVSS-Share$_j$ then check whether $g_i(j) = f_j(i)$. When the test passes for all $j = 1, \ldots, n$, then A-cast `Matched-Column`.

iii. CCORE Construction: Code for D – Only D executes this code.

1. For $i = 1, \ldots, n$, construct $VCORE$ for SAVSS-Share($D, \mathcal{P}, f_i(x), \epsilon'$). Denote it by $VCORE^i$.
2. For $i = 1, \ldots, n$, keep updating $VCORE^i$ even after $|VCORE^i| = 2t + 1$. Wait to obtain $CCORE = \cap_{i=1}^{n} VCORE^i$ of size at least $2t + 1$ such that `Matched-Column` is received from A-cast of every $P_j \in CCORE$.
3. A-cast $CCORE$.

iv. CCORE Verification & Agreement: Code for P_i

1. Wait to receive $CCORE$ from the A-cast of D.
2. Check whether $CCORE$ is a valid $VCORE$ for SAVSS-Share$_j$ for every $j = 1, \ldots, n$ (by following the steps 2-4 as specified under [**VCORE Verification & Agreement on VCORE: Code for P_i**] in code Re-commitment of SAVSS-Share). If yes then wait to receive `Matched-Column` from A-cast of every $P_j \in CCORE$ and then accept $CCORE$.

v. P_j-private-reconstruction of $f_j(x)$ for $j = 1, \ldots, n$: Code for P_i

1. If CCORE is a valid $VCORE$ for SAVSS-Share$_j$ for every $j = 1, \ldots, n$, then participate in SAVSS-Rec-Private($D, \mathcal{P}, f_j(x), P_j, \epsilon'$), for $j = 1, \ldots, n$, to enable P_j-private-reconstruction of $f_j(x)$. We refer SAVSS-Rec-Private($D, \mathcal{P}, f_j(x), P_j, \epsilon'$) as SAVSS-Rec-Private$_j$. Notice that CCORE is used as VCORE in each SAVSS-Rec-Private$_j$, for $j = 1, \ldots, n$.
2. At the completion of SAVSS-Rec-Private$_i$, obtain degree-t polynomial $f_i(x)$.
3. Assign $s_i = f_i(0)$. Output s_i as i^{th} share of s and terminate ACSS-Share.

Figure 10.2: Protocol ACSS-Rec-Private and ACSS-Rec-Public for Reconstructing Secret s privately and publicly (respectively) with $n = 3t + 1$

ACSS-Rec-Private($D, \mathcal{P}, s, P_\alpha, \epsilon$): P_α-private-reconstruction of s:

i. Code for P_i: – Every party in $\mathcal{P}$ executes this code.

1. Privately send s_i, the i^{th} share of s to P_α.

ii. Code for P_α: – Only $P_\alpha \in \mathcal{P}$ executes this code.

1. Apply OEC on received shares of s to reconstruct s and terminate ACSS-Rec-Private.

ACSS-Rec-Public($D, \mathcal{P}, s, \epsilon$): Public reconstruction of s:

i. Code for P_i – Every party in $\mathcal{P}$ executes this code.

1. Privately send s_i, the i^{th} share of s to every party $P_j \in \mathcal{P}$.
2. Apply OEC on received shares of s to reconstruct s and terminate ACSS-Rec-Public.

1. *If D is honest then eventually he will generate a $CCORE$ of size $2t + 1$ except with probability ϵ. Moreover, each honest party will eventually agree on $CCORE$.*

2. *If D is corrupted and some honest party has accepted the $CCORE$ received from the A-cast of D, then every other honest party will eventually accept $CCORE$.*

Proof: In ACSS-Share if D is honest then from the proof of Lemma 8.34, an *honest* party may be added in each $VCORE^i$ except with probability $\epsilon' = \frac{\epsilon}{n}$ (recall that each instance of SAVSS-Share has an associated error probability of $\epsilon' = \frac{\epsilon}{n}$). So even though there are no common corrupted parties among $VCORE^i$'s, eventually all the honest parties will be common among n $VCORE^i$'s with probability at least $1 - (2t + 1)\epsilon' \approx 1 - \epsilon$. Moreover, each honest P_i will eventually A-cast `Matched-Column` signal, as $f_j(i) = g_i(j)$ will hold for all $j = 1, \ldots, n$ when D is *honest*. It may be possible that some corrupted parties are also added in each $VCORE^i$. Moreover those corrupted parties may even A-cast `Matched-Column` signal. So except with probability ϵ, at some point of time $CCORE = \cap_{i=1}^{n} VCORE^i$ will contain at least $2t + 1$ parties who have A-casted `Matched-Column` signal. So honest D will find $CCORE$ and A-cast the same. Now it is easy to see that each honest party will accept $CCORE$ after receiving it from A-cast of D and verifying its' validity after following steps in iv(2) of protocol ACSS-Share.

If D is corrupted and some honest party, say P_i has accepted $CCORE$ received from the A-cast of D, then P_i must have checked the condition specified in iv(2) of protocol ACSS-Share. The same will hold for all other honest parties who will eventually accept $CCORE$. □

Lemma 10.5 *In ACSS-Share, if the honest parties agree on* CCORE*, then it implies that D is committed to a unique degree-(t,t) bivariate polynomial $\overline{F}(x,y)$ such that each row polynomial $f_i(x)$ committed by D in SAVSS-Share$_i$ satisfies $\overline{F}(x,i) = f_i(x)$ and the column polynomial $g_j(y)$ held by every honest $P_j \in$ CCORE satisfies $\overline{F}(j,y) = g_j(y)$. Moreover if D is honest then $\overline{F}(x,y) = F(x,y)$.*

Proof: The proof follows from the same argument as given in Lemma 8.31. □

Lemma 10.6 *In ACSS-Share, if the honest parties agree on* CCORE*, then eventually all honest parties will get their share of D's committed secret $\overline{s}$, except with probability at most ϵ. That is, protocol ACSS-Share will generate t-$(1d)$-sharing of $\overline{s}$ except with probability ϵ. Moreover if D is honest then $\overline{s} = s$.*

Proof: From the previous lemma, if the honest parties agree on CCORE then it implies that D is committed to a unique degree-(t,t) bivariate polynomial $\overline{F}(x,y)$ such that each row polynomial $f_i(x)$ committed by D in SAVSS-Share$(D, \mathcal{P}, f_i(x), \epsilon')$ satisfies $\overline{F}(x,i) = f_i(x)$. Now from the properties of SAVSS-Rec-Private, P_i-private-reconstruction of $f_i(x)$ will enable honest P_i to obtain $f_i(x)$ and hence his share $f_i(0)$, except with probability ϵ'. As there are $2t+1$ honest parties in $\mathcal{P}$, all honest parties will obtain their share of the secret $\overline{s} = \overline{F}(0,0)$, except with probability $(2t+1)\epsilon' \approx \epsilon$. Hence the secret $\overline{s} = \overline{F}(0,0)$ will be t-shared by the degree-t polynomial $\overline{F}(x,0)$, except with probability at most ϵ. □

Lemma 10.7 (ACSS-Termination) *Protocol ACSS satisfies* **termination** *property of Definition 10.2.*

Proof: **Termination 1** and **Termination 2** follows from Lemma 10.4 and Lemma 10.6. **Termination 3** follows from Lemma 10.6 and properties of OEC. □

Lemma 10.8 (ACSS-Secrecy) *Protocol ACSS satisfies* **secrecy** *property of Definition 10.2.*

Proof: Here we have to consider the case when D is honest. Without loss of generality, assume that $P_1, \ldots, P_t$ are the parties under the control of $\mathcal{A}_t$. So during ACSS-Share, $\mathcal{A}_t$ will know $f_1(x), \ldots, f_t(x), g_1(y), \ldots, g_t(y)$ and t points on $f_{t+1}(x), \ldots, f_n(x)$. From the secrecy property of SAVSS-Share (Lemma 8.37), $\mathcal{A}_t$ will have no information about $f_{t+1}(0), \ldots, f_n(0)$ during the execution of corresponding instances of SAVSS-Share. So from the properties of bi-variate polynomial of degree-(t,t) [46], the adversary $\mathcal{A}_t$ will lack one more point to uniquely interpolate $F(x,y)$ during ACSS-Share. Hence hence the secret $s = F(0,0)$ will remain information theoretically secure from $\mathcal{A}_t$. □

Lemma 10.9 (ACSS-Correctness) *Protocol ACSS satisfies* **correctness** *property of Definition 10.2.*

Proof: Follows from Lemma 10.4, Lemma 10.5, Lemma 10.6 and from the properties of OEC. □

Lemma 10.10 (ACSS-Completeness) *Protocol ACSS satisfies* **completeness** *property of Definition 10.2.*

PROOF: Follows from Lemma 10.6. □

Theorem 10.11 *Protocol ACSS consisting of (SAVSS-Share, ACSS-Rec-Private, AVSS-Rec-Public) constitutes a valid statistical ACSS scheme for sharing a single secret.*

PROOF: Follows from Lemma 10.7, Lemma 10.9, Lemma 10.8 and Lemma 10.10. □

Theorem 10.12 (Communication Complexity of ACSS)

- *Protocol ACSS-Share privately communicates $\mathcal{O}(n^5(\log \frac{1}{\epsilon})^2)$ bits and A-casts $\mathcal{O}(n^4 \log n)$ bits.*
- *Protocol ACSS-Rec-Private incurs a private communication of $\mathcal{O}(n \log \frac{1}{\epsilon})$ bits.*
- *Protocol ACSS-Rec-Public incurs a private communication of $\mathcal{O}(n^2 \log \frac{1}{\epsilon})$ bits.*

PROOF: The communication complexity of ACSS-Share follows from Theorem 8.42 (that states the communication complexity of SAVSS) and the fact that in ACSS-Share, there are n executions of SAVSS-Share and SAVSS-Rec-Private, each with an error parameter of $\frac{\epsilon}{n}$. In ACSS-Rec-Private, each party sends his share to P_α, incurring a total communication cost of $\mathcal{O}(n \log \frac{1}{\epsilon})$ bits. In ACSS-Rec-Public, each party sends his share to every other party, incurring a total communication cost of $\mathcal{O}(n^2 \log \frac{1}{\epsilon})$ bits. □

10.2.3 Statistical ACSS for Sharing Multiple Secrets

We now present an ACSS scheme ACSS-MS, consisting of sub-protocols (ACSS-MS-Share, ACSS-MS-Rec-Private, ACSS-MS-Rec-Public). Protocol ACSS-MS-Share allows D to generate t-$(1d)$-sharing of secret $S = (s^1, \ldots, s^\ell)$, consisting of $\ell > 1$ elements from $\mathbb{F}$. While D can ACSS-share S using ℓ executions of ACSS-Share, one for each $s^l \in S$, with a private communication of $\mathcal{O}((\ell n^5 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(\ell n^4 \log(n))$ bits, protocol ACSS-MS-Share achieves the same task with a private communication of $\mathcal{O}((\ell n^4 + n^5 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^4 \log(n))$ (independent of ℓ) bits. This shows that executing a *single instance* of ACSS-MS dealing with *multiple secrets* concurrently is advantageous over executing *multiple instances* of ACSS dealing with *single secret*. Protocol ACSS-MS-Share is provided in Fig. 10.3. Protocol ACSS-MS-Rec-Private and ACSS-MS-Rec-Public are presented in Fig. 10.4.

Protocol ACSS uses the strong statistical AVSS protocol called SAVSS-MS (consisting of sub-protocols (SAVSS-MS-Share, SAVSS-MS-Rec-Private)) for sharing multiple secrets, presented in Sections 8.5.3 and 8.5.4 of Chapter 8.

To bound the error probability by ϵ, the computation of ACSS-MS is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^5 \kappa 2^{-\kappa}$. This is derived from the fact that in ACSS-MS, protocol SAVSS-MS is invoked with $\frac{\epsilon}{n}$ error probability and as mentioned in Section 8.5.4, $\epsilon \geq n^4 \kappa 2^{-\kappa}$ should hold to bound error probability of SAVSS-MS by ϵ. So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

The proof of the properties of ACSS-MS can be directly extended from the proof of the properties of ACSS.

Figure 10.3: Protocol ACSS-MS-Share for Sharing Secret S Containing ℓ Elements with $n = 3t + 1$

Protocol ACSS-MS-Share($D, \mathcal{P}, S, \epsilon$)

i. Distribution by D: Code for D: – Only D executes this code

1. Select ℓ random degree-(t,t) bivariate polynomials $F^1(x,y), \ldots, F^\ell(x,y)$ such that $F^l(0,0) = s^l$ for $l = 1, \ldots, \ell$.
2. For $i = 1, \ldots, n$, send $g_i^l(y) = F^l(i,y)$ for $l = 1, \ldots, \ell$ to P_i. We call polynomials $g_i^1(y), \ldots, g_i^\ell(y)$ as i^{th} column polynomials.
3. For $i = 1, \ldots, n$, initiate SAVSS-MS-Share$(D, \mathcal{P}, (f_i^1(x), \ldots, f_i^\ell(x)), \epsilon')$ (See Notation 8.45 for the syntax) for sharing $(f_i^1(x), \ldots, f_i^\ell(x))$, where $f_i^l(x) = F^l(x,i)$ and $\epsilon' = \frac{\epsilon}{n}$. We call polynomials $f_i^1(x), \ldots, f_i^\ell(x)$ as i^{th} row polynomials. We refer SAVSS-MS-Share$(D, \mathcal{P}, (f_i^1(x), \ldots, f_i^\ell(x)), \epsilon')$ as SAVSS-MS-Share$_i$

ii. Code for P_i: – Every party in $\mathcal{P}$, including D, executes this code

1. Wait to receive degree-t polynomials $g_i^l(y)$ for $l = 1, \ldots, \ell$ from D.
2. Participate in SAVSS-MS-Share$_j$ for all $j = 1, \ldots, n$.
3. If $(f_j^1(i), \ldots, f_j^\ell(i))$ is received from D during SAVSS-MS-Share$_j$ then check whether $g_i^l(j) = f_j^l(i)$ holds for all $l = 1, \ldots, \ell$. When the test passes for all $j = 1, \ldots, n$, then A-cast `Matched-Column`.

iii. CCORE Construction: Code for D – Only D executes this code.

1. For $i = 1, \ldots, n$, construct $VCORE$ for SAVSS-MS-Share$_i$. Denote it by $VCORE^i$.
2. For $i = 1, \ldots, n$, keep updating $VCORE^i$, even after $|VCORE^i| = 2t + 1$. Wait to obtain $CCORE = \cap_{i=1}^n VCORE^i$ of size at least $2t + 1$ such that `Matched-Column` is received from A-cast of every $P_j \in CCORE$.
3. A-cast $CCORE$.

iv. CCORE Verification & Agreement: Code for P_i – Every party including D will execute this code.

1. Wait to receive $CCORE$ from the A-cast of D.
2. Check whether $CCORE$ is a valid $VCORE$ for SAVSS-MS-Share$_j$ for every $j = 1, \ldots, n$ (by following the steps 2-4 as specified under [**VCORE Verification & Agreement on VCORE: Code for** P_i] in protocol Re-commitment-MS).

v. P_j-private-reconstruction of $(f_j^1(x), \ldots, f^\ell(x))$ for $j = 1, \ldots, n$: Code for P_i

1. If CCORE is a valid $VCORE$ for SAVSS-MS-Share$_j$ for every $j = 1, \ldots, n$, then participate in SAVSS-MS-Rec-Private$(D, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), P_j, \epsilon')$, for $j = 1, \ldots, n$, to enable P_j-private-reconstruction of $(f_j^1(x), \ldots, f_j^\ell(x))$. We refer SAVSS-MS-Rec-Private$(D, \mathcal{P}, (f_j^1(x), \ldots, f_j^\ell(x)), P_j, \epsilon')$ by SAVSS-MS-Rec-Private$_j$. Notice that CCORE is used as VCORE in each SAVSS-MS-Rec-Private$_j$, for $j = 1, \ldots, n$.
2. At the completion of SAVSS-MS-Rec-Private$_i$, obtain degree-t polynomials $(f_i^1(x), \ldots, f_i^\ell(x))$.
3. Assign $s_i^l = f_i^l(0)$. Output $(s_i^1, \ldots, s_i^\ell)$ as i^{th} share of $(s^1, \ldots, s^\ell)$ and terminate ACSS-MS-Share.

Theorem 10.13 *Protocol ACSS-MS consisting of sub-protocols (ACSS-MS-Share, ACSS-MS-Rec-Private, ACSS-MS-Rec-Public) is a valid statistical ACSS scheme for sharing $\ell \geq 1$ secrets.*

Theorem 10.14 (Communication Complexity of ACSS-MS)

Figure 10.4: Protocol ACSS-MS-Rec-Private and ACSS-MS-Rec-Public for Reconstructing Secret S privately and publicly (respectively) with $n = 3t + 1$

ACSS-MS-Rec-Private($D, \mathcal{P}, S, P_\alpha, \epsilon$): P_α-private-reconstruction of S:

i. Code for P_i: – Every party in $\mathcal{P}$ executes this code.

1. Privately send $s_i^1, \ldots, s_i^\ell$, the i^{th} shares of $s^1, \ldots, s^\ell$ (respectively) to party $P_\alpha \in \mathcal{P}$.

ii. Code for P_α: – Only $P_\alpha \in \mathcal{P}$ executes this code.

1. For $l = 1, \ldots, \ell$, apply OEC on received shares of s^l to reconstruct s^l and terminate ACSS-MS-Rec-Private.

ACSS-MS-Rec-Public($D, \mathcal{P}, S, \epsilon$): Public reconstruction of S:

i. Code for P_i: – Every party in $\mathcal{P}$ executes this code.

1. Privately send $s_i^1, \ldots, s_i^\ell$, the i^{th} shares of $s^1, \ldots, s^\ell$ (respectively) to every party $P_j \in \mathcal{P}$.
2. For $l = 1, \ldots, \ell$, apply OEC on received shares of s^l to reconstruct s^l and terminate ACSS-MS-Rec-Public.

- *Protocol ACSS-MS-Share privately communicates* $\mathcal{O}((\ell n^4 + n^5 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ *bits and A-casts* $\mathcal{O}(n^4 \log n)$ *bits.*
- *Protocol ACSS-MS-Rec-Private incurs a private communication of* $\mathcal{O}(\ell n \log \frac{1}{\epsilon})$ *bits.*
- *Protocol ACSS-MS-Rec-Public incurs a private communication of* $\mathcal{O}(\ell n^2 \log \frac{1}{\epsilon})$ *bits.*

PROOF: The communication complexity of ACSS-MS-Share follows from Theorem 8.46 (that states the communication complexity of SAVSS-MS) and from the fact that $\mathcal{O}(n)$ instances of SAVSS-MS-Share and SAVSS-MS-Rec-Private each with ℓ secrets may be executed in ACSS-MS-Share.

In ACSS-MS-Rec-Private, each party sends his shares of ℓ secrets to P_α, incurring a total communication cost of $\mathcal{O}(\ell n \log \frac{1}{\epsilon})$ bits. In ACSS-Rec-Public, each party sends his shares of ℓ secrets to every other party, incurring a total communication cost of $\mathcal{O}(\ell n^2 \log \frac{1}{\epsilon})$ bits. □

Notation 10.15 (Notation for Using ACSS-MS) *In the subsequent sections, we will invoke ACSS-MS-Share as ACSS-MS-Share* $(D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x)), \epsilon)$ *to mean that D commits to $f^1(x), \ldots, f^\ell(x)$ in ACSS-MS-Share. Essentially here D is asked to choose bivariate polynomials $F^1(x, y), \ldots, F^\ell(x, y)$, each of degree-$(t, t)$, such that $F^l(x, 0) = f^l(x)$ holds for $l = 1, \ldots, \ell$. As a result of this execution, each honest party P_i will get the shares $f^1(i), \ldots, f^\ell(i)$. Similarly, ACSS-MS-Rec-Private will be invoked as ACSS-MS-Rec-Private*$(D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x)), P_\alpha, \epsilon)$

to enable $P_\alpha \in \mathcal{P}$ to privately reconstruct $(f^1(x), \ldots, f^\ell(x))$. Similarly, ACSS-MS-Rec-Public *will be invoked as* ACSS-MS-Rec-Public($D, \mathcal{P}, (f^1(x), \ldots, f^\ell(x)), \epsilon$) *to enable each party in $\mathcal{P}$ to reconstruct $(f^1(x), \ldots, f^\ell(x))$.*

10.3 Primitives Used in Our AMPC Protocol

In addition to the ACSS scheme proposed by us, our AMPC protocol also uses a well known primitive called *Agreement on Common Subset* (ACS) [13, 21].

Agreement on Common Subset (ACS) [13, 21]: It is an asynchronous primitive presented in [19, 21]. It outputs a common set, containing at least $n-t$ parties, who correctly shared their values. Moreover, each honest party will eventually get a share, corresponding to each value, shared by the parties in the common set. Actually, ACS calls n instances of ABA protocol. To maintain an error probability of ϵ, each ABA should be invoked with $\frac{\epsilon}{n}$ error probability. So, if we consider our ABA presented in Chapter 9, then this requires a total communication complexity of $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits of A-cast or $\mathcal{O}(n^8 \log \frac{1}{\epsilon})$ bits private communication (This is because A-cast of ℓ bit message requires ℓn^2 bits of private communication; see Theorem 7.2). Also ACS protocol has to work on field $\mathbb{F} = GF(2^\kappa)$ where κ has to be determined using the relation $\epsilon \geq 4n^7 2^{-\kappa}$. This is because our ABA enforces $\epsilon \geq 4n^6 2^{-\kappa}$ to maintain error probability ϵ.

In our AMPC protocol, we use another simple protocol RNG very frequently, that allows the parties in $\mathcal{P}$ to jointly generate a random, non-zero element $r \in \mathbb{F}$. The protocol uses our ACSS scheme and the ACS protocol as black-boxes.

Random Number Generation (RNG): The protocol for random number generation works as follows: each $P_i \in \mathcal{P}$ shares a random non-zero $r_i \in \mathbb{F}$ using ACSS-Share with an error probability $\frac{\epsilon}{n}$. The parties then run ACS with error parameter ϵ to agree on a common set, say $\mathcal{C}$ of at least $2t+1$ parties who did proper sharing of their random values. Once $\mathcal{C}$ is agreed upon, ACSS-Rec-Public is executed for every $P_i \in \mathcal{C}$ in order to reconstruct back P_i's committed secret. Now every party in $\mathcal{P}$ locally add the committed secret of every $P_i \in \mathcal{C}$. It is easy to see that the sum value is random. We call this protocol as RNG. Protocol RNG will have an error probability of ϵ. The protocol privately communicates $\mathcal{O}(n^6 (\log \frac{1}{\epsilon})^2)$ bits (because of ACSS) and A-casts $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits (because of ACS). Moreover RNG needs to work on a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq \max(4n^7 2^{-\kappa}, n^6 \kappa 2^{-\kappa}) \Rightarrow \epsilon \geq n^6 2^{-\kappa} \max(4n, \kappa)$. This is because of ACS and ACSS.

10.4 The Approach Used in the AMPC of [21] and Current Chapter

AMPC of [21]: The AMPC protocol of [21] consists of input phase and computation phase. In input phase every party P_i shares (or commits to) his input x_i. All the parties then decide on a common set of $n-t$ parties (using ACS) who have done proper sharing of their input. Once this is done, in the computation phase the arithmetic circuit representing f is computed gate by gate, such that the intermediate gate outputs are always kept as secret and are properly shared among the parties, following the approach of [20]. Now for sharing/committing inputs, a natural choice is to use AVSS protocol which can be treated as a form of *commitment*, where the commitment is held in a distributed fashion among the parties. Before [21], the only known statistical AVSS scheme with $n = 3t+1$ was due to [39]. But it is shown in [21] that the use of the AVSS protocol of [39] for committing inputs (secrets), does not allow to compute the circuit robustly in a straight-forward way. This is because *for robust computation of the circuit, it is to be ensured that at the end of AVSS sharing phase, every honest party should have access to share of the secret*. Unfortunately the AVSS of [39] does not guarantee the above property, which we may refer as *ultimate* property. This very reason motivated Ben-Or et al. [21] to introduce a new asynchronous primitive called *Ultimate Secret Sharing* (USS) which not only ensures that every honest party has access to his share of the secret, but also offers all the properties of AVSS. Thus [21] presents an USS scheme with $n = 3t+1$ using the AVSS protocol of [39] as a building block. Essentially, in the USS protocol of [21], every share of the secret is committed using AVSS of [39] which ensures that each honest party P_i can have an access to the i^{th} share of secret by means of private reconstruction of AVSS. A secret s that is shared using USS is called *ultimately shared*. Now in the input phase of AMPC in [21], parties *ultimately share* their inputs. Then in the computation phase, for every gate (except output gate), *ultimate sharing* of the output is computed from the *ultimate sharing* of the inputs, following the approach of [20, 138].

Now we carry out the communication complexity analysis for the AMPC of [21]. The analysis of communication complexity of the AVSS of [39, 35] is performed in section 8.7 of Chapter 8. The sharing phase of the AVSS involves a private communication of $\Omega(n^9(\log\frac{1}{\epsilon})^4)$ bits and $\mathsf{A\text{-}cast}$ of $\Omega(n^9(\log\frac{1}{\epsilon})^2\log(n))$ bits, for sharing a single secret. As the sharing phase of the USS scheme of [21] requires n invocations to the sharing phase of AVSS of [39], it incurs a private communication of $\Omega(n^{10}(\log\frac{1}{\epsilon})^4)$ bits and $\mathsf{A\text{-}cast}$ of $\Omega(n^{10}(\log\frac{1}{\epsilon})^2\log(n))$ bits. Finally in the AMPC protocol, each multiplication requires n invocations to the sharing phase of USS. So evaluation of each multiplication gate incurs a private communication of $\Omega(n^{11}(\log\frac{1}{\epsilon})^4)$ and $\mathsf{A\text{-}cast}$ of $\Omega(n^{11}(\log\frac{1}{\epsilon})^2\log(n))$ bits.

AMPC of Current Chapter: Our statistical AMPC protocol follows the preprocessing model of [5] and proceeds in a sequence of three phases: preparation phase, input phase and computation phase. Every honest party will eventually complete each phase with very high probability. We call a triple (a, b, c) as a random multiplication triple if a, b are random and $c = ab$. In the preparation phase, t-$(1d)$-sharing of $c_M + c_R$ random multiplication triples are generated. Each multiplication and random gate of the circuit is associated with a multiplication triple. In the input phase the parties t-$(1d)$-share (commit to) their inputs and then agree on a common subset of $n-t$ parties (using ACS) who correctly shared their inputs. In the computation phase, the actual circuit will be computed gate by gate, based on the inputs of the parties in common set. Due to the linearity of the used secret-sharing, the linear gates can be computed locally. Each multiplication gate will be evaluated using the circuit randomization technique of [5] with the help of the associated multiplication triple (generated in preparation phase).

For committing/sharing secrets, we use our ACSS scheme. There is a slight *definitional difference* between the USS of [21] and our ACSS, though both of them offer all the properties of AVSS. While USS of [21] ensures that *every* honest party has *access* to share of secret (*but may not hold the share directly*), our ACSS ensures that *every honest party holds his share* of secret. This property of ACSS is called *completeness* property as mentioned in the definition of ACSS (see Definition 10.2). The advantages of ACSS over USS are as follows:

1. It makes the computation of the gates very simple;
2. Reconstruction phase of ACSS is very simple, efficient and can be achieved using OEC of [35].

Apart from these advantages, our ACSS is strikingly better than USS of [21] in terms of communication complexity. While sharing phase of our ACSS privately communicates $\mathcal{O}((\ell n^4 + n^5 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-casts $\mathcal{O}(n^4 \log n)$ bits to share ℓ *secrets concurrently*, the sharing phase of USS in [21] privately communicates $\Omega(n^{10}(\log \frac{1}{\epsilon})^4)$ bits and A-casts $\Omega(n^{10}(\log \frac{1}{\epsilon})^2 \log(n))$ bits to share *only one secret*.

Prior to our discussion on AMPC protocol, we design another important protocol for generating t-$(2d)$-sharing (the definition of t-$(2d)$-sharing has been presented in Definition 6.16) that uses our ACSS scheme as a building block. This protocol will also be used as building block in our AMPC protocol.

10.5 Generating t-$(2d)$-Sharing

From the definition of t-$(2d)$-sharing (see Definition 6.16), we see that the t-$(2d)$-sharing of s implies that s as well as its shares are individually t-$(1d)$-shared. Now we present a protocol t-(2d)-Share which allows a special party $D \in \mathcal{P}$ to simultaneously generate t-$(2d)$-sharing of $\ell \geq 1$ elements from $\mathbb{F}$, namely $s^1, \ldots, s^\ell$. In protocol t-(2d)-Share, the following happen with probability at least $(1 - \epsilon)$: (a) If D is honest, then every honest party will eventually terminate t-(2d)-Share; (b) Moreover, if D is corrupted and some honest party has terminated t-(2d)-Share, then eventually every other honest party will also terminate t-(2d)-Share; (c) Furthermore, if some honest party has terminated t-(2d)-Share, then it implies that D has done correct t-$(2d)$-sharing of $s^1, \ldots, s^\ell$.

The intuition: The high level idea of the protocol is as follows: D selects a random value $s^0 \in \mathbb{F}$ and hides each s^i (where $i = 0, \ldots, \ell$) in the constant term of a random degree-t polynomial $q^i(x)$. D then t-$(1d)$-shares the secret $S^0 = (s^0, \ldots, s^\ell)$, as well as their i^{th} shares $S^i = (q^0(i), \ldots, q^\ell(i))$. The parties then jointly employ a verification technique to ensure that D indeed t-$(1d)$-shared S^i for $i = 1, \ldots, n$ which are the shares of S^0. A similar verification technique was used in [12] in synchronous settings. The secret s^0 is used to ensure the secrecy of $s^1, \ldots, s^\ell$ during the verification process. After verification, the polynomials used for t-$(1d)$-sharing S_i are privately reconstructed by P_i, thus completing the t-$(2d)$-sharing of $s^1, \ldots, s^\ell$. The above idea is implemented in protocol t-(2d)-Share which is given in Fig. 10.5.

To bound the error probability by ϵ, the computation of t-(2d)-Share is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^6 2^{-\kappa} \max(4n, \kappa)$. This is derived from the fact that in t-(2d)-Share,

1. ACSS-MS is invoked with $\frac{\epsilon}{n+1}$ error probability and as mentioned in Section 10.2.3, $\epsilon \geq n^5 \kappa 2^{-\kappa}$ should hold to bound error probability of ACSS-MS by ϵ and

2. RNG is invoked with error probability ϵ and this enforces $\epsilon \geq n^6 2^{-\kappa} \max(4n, \kappa)$.

So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.
We now prove the properties of protocol $\mathsf{t\text{-}(2d)\text{-}Share}$.

Lemma 10.16 *Protocol $\mathsf{t\text{-}(2d)\text{-}Share}$ satisfies the following properties:*

1. **Termination:** *If D is honest, then except with probability ϵ, all honest parties will eventually terminate $\mathsf{t\text{-}(2d)\text{-}Share}$. Moreover if some honest party has terminated $\mathsf{t\text{-}(2d)\text{-}Share}$, then every honest party will eventually terminate $\mathsf{t\text{-}(2d)\text{-}Share}$, except with probability ϵ.*

2. **Correctness:** *If some honest party has terminated the protocol then except with probability ϵ, it is ensured that D has done correct t-$(2d)$-sharing of $s^1, \ldots, s^\ell$.*

3. **Secrecy:** *If D is honest then $s^1, \ldots, s^\ell$ will remain secure.*

PROOF: **Termination:** When D is honest, every $\mathsf{ACSS\text{-}MS\text{-}Share}_i$ initiated by honest D will terminate with its desired output (t-$(1d)$-sharings) except with probability ϵ'. Therefore all the $n+1$ instances of $\mathsf{ACSS\text{-}MS\text{-}Share}_i$ will terminate with their desired output (i.e., t-$(1d)$-sharing of $S^0, \ldots, S^n$), except with probability $\epsilon'(n+1) \approx \epsilon$. Since t-$(1d)$-sharing of $S^0, \ldots, S^n$ are generated properly, the verification steps specified in $\mathsf{t\text{-}(2d)\text{-}Share}$ will pass. Subsequently, n instances of $\mathsf{ACSS\text{-}MS\text{-}Rec\text{-}Private}$ are executed in order to complete t-$(2d)$-sharing of S. Due to the property of $\mathsf{ACSS\text{-}MS\text{-}Rec\text{-}Private}$, each honest P_i will correctly reconstruct the required information corresponding to t-$(2d)$-sharing of S, namely $q_i^0(x), \ldots, q_i^\ell(x)$ and will terminate $\mathsf{t\text{-}(2d)\text{-}Share}$ except with probability ϵ'. As there are at least $2t+1$ honest parties, except with probability $(2t+1)\epsilon' \approx \epsilon$, all honest parties will eventually terminate $\mathsf{t\text{-}(2d)\text{-}Share}$ with correct t-$(2d)$-sharing of $s^1, \ldots, s^\ell$. This completes the proof of the first part of **Termination** property. We now proceed to prove second part of the **Termination** property.

Let P_i be an honest party who has terminated protocol $\mathsf{t\text{-}(2d)\text{-}Share}$. This implies that P_i has reconstructed $q_i^0(x), \ldots, q_i^\ell(x)$ from $\mathsf{ACSS\text{-}MS\text{-}Rec\text{-}Private}_i$. This further means there is at least one honest party who checked that the public verification has passed and participated in every $\mathsf{ACSS\text{-}MS\text{-}Rec\text{-}Private}_j$ for $j = 1, \ldots, n$. As the verification process is public, every other honest party will eventually see that the verification passes and then they will participate in $\mathsf{ACSS\text{-}MS\text{-}Rec\text{-}Private}_j$ for $j = 1, \ldots, n$. Now by the termination property of $\mathsf{ACSS\text{-}MS\text{-}Rec\text{-}Private}$, all honest P_js will terminate $\mathsf{ACSS\text{-}MS\text{-}Rec\text{-}Private}_j$, output $q_j^0(x), \ldots, q_j^\ell(x)$ and finally terminate protocol $\mathsf{t\text{-}(2d)\text{-}Share}$, except with probability $n\epsilon' \approx \epsilon$.

Correctness: Here we consider two cases: (a) when D is honest; (b) when D is corrupted.

1. When D is honest, then correctness follows from the proof of the first part of termination property.

2. Now we consider D to be corrupted. For $i = 0, \ldots, n$, $\mathsf{ACSS\text{-}MS\text{-}Share}_i$ ensures that D has correctly t-$(1d)$-shared $\overline{S^i}$, except with probability ϵ' (by **Completeness** property of $\mathsf{ACSS\text{-}MS}$). But it may happen that $\overline{S^i}$ which is t-$(1d)$-shared by D in $\mathsf{ACSS\text{-}MS\text{-}Share}_i$, is not the correct i^{th} shares of S^0.

Figure 10.5: Protocol t-(2d)-Share for Generating t-$(2d)$-sharing of $S = (s^1, \ldots, s^\ell)$, $n = 3t + 1$

Protocol t-(2d)-Share($D, \mathcal{P}, S, \epsilon$)

Sharing by D: Code for D — Only D executes this code

1. Select a random s^0 and $\ell + 1$ degree-t random polynomials $q^0(x), \ldots, q^\ell(x)$ such that for $l = 0, \ldots, \ell$, $q^l(0) = s^l$. Let $s_i^l = q^l(i)$ and $S^i = (q^0(i), \ldots, q^\ell(i))$ for $i = 0, \ldots, n$. So $S^0 = (s^0, \ldots, s^\ell)$ and $S^i = (s_i^0, \ldots, s_i^\ell)$.
2. For $l = 0, \ldots, \ell$ and $i = 1, \ldots, n$, select random degree-t polynomials $q_i^l(x)$, such that $q_i^l(0) = q^l(i) = s_i^l$. Let $S^{ij} = (q_i^0(j), q_i^1(j), \ldots, q_i^\ell(j)) = (s_{ij}^0, s_{ij}^1, \ldots, s_{ij}^\ell)$, for $j = 1, \ldots, n$.
3. Invoke ACSS-MS-Share($D, \mathcal{P}, (q^0(x), q^1(x), \ldots, q^\ell(x)), \epsilon'$), where $\epsilon' = \frac{\epsilon}{n+1}$, for generating t-$(1d)$-sharing of S^0. Denote this instance of ACSS-MS-Share by ACSS-MS-Share$_0$. During ACSS-MS-Share$_0$, party P_j receives the shares S^j for $j = 1, \ldots, n$.
4. For $i = 1, \ldots, n$, invoke ACSS-MS-Share($D, \mathcal{P}, (q_i^0(x), q_i^1(x), \ldots, q_i^\ell(x)), \epsilon'$), where $\epsilon' = \frac{\epsilon}{n+1}$, for generating t-$(1d)$-sharing of S^i. Denote this instance of ACSS-MS-Share by ACSS-MS-Share$_i$. During ACSS-MS-Share$_i$, party P_j receives the share-shares S^{ij}, for $j = 1, \ldots, n$.

Verification: Code for P_i — Every party in $\mathcal{P}$ executes this code

1. Upon completion of ACSS-MS-Share$_j$ for all $j \in \{0, \ldots, n\}$, participate in protocol RNG to generate random r with error probability ϵ.
2. Wait to terminate RNG with r as output. Compute $s_i^* = \sum_{l=0}^{\ell} r^l s_i^l$ which is the i^{th} share of $s^* = \sum_{l=0}^{\ell} r^l s^l$. In addition, for $j = 1, \ldots, n$, locally compute $s_{ji}^* = \sum_{l=0}^{\ell} r^l s_{ji}^l$ which is the i^{th} share-share of s_j^*.
3. Participate in ACSS-MS-Rec-Public($D, \mathcal{P}, (s^*, s_1^*, \ldots, s_n^*), \epsilon$) to publicly reconstruct $s^*, s_1^*, \ldots, s_n^*$. This results in every party reconstructing $q^*(x)$ and $q_1^*(x), \ldots, q_n^*(x)$ with $q^*(0) = s^*$ and $q_i^*(0) = s_i^*$.
4. Check whether for $i = 1, \ldots, n$, $q^*(i) \stackrel{?}{=} q_i^*(0)$. If yes proceed to the next step assuming that D has done proper t-$(1d)$-sharing of S^j for $j = 0, \ldots, n$.

Private reconstruction of polynomials used for sharing S^j by P_j: Code for P_i — Every party in $\mathcal{P}$ executes this code

1. For $j = 1, \ldots, n$, participate in ACSS-MS-Rec-Private($D, \mathcal{P}, S^j, P_j, \epsilon'$) for enabling P_j to privately reconstruct the polynomials $q_j^0(x), \ldots, q_j^\ell(x)$ which were used by D to share S^j. We refer ACSS-MS-Rec-Private($D, \mathcal{P}, S^j, P_j, \epsilon'$) by ACSS-MS-Rec-Private$_j$.
2. Wait to privately reconstruct degree-t polynomials $q_i^0(x), \ldots, q_i^\ell(x)$ from ACSS-MS-Rec-Private$_i$ and terminate t-(2d)-Share.

Assume that D has t-$(1d)$-shared $\overline{S^j} \neq S^j$ in ACSS-MS-Share$_j$, for some $j \in \{1, \ldots, n\}$. This implies that in ACSS-MS-Share$_j$, D has used polynomials $\overline{q_j^0}(x), \ldots, \overline{q_j^\ell}(x)$ to share $\overline{S^j}$, such that for at least one $l \in \{0, \ldots, \ell\}$, $\overline{q_j^l}(0) \neq q^l(j) = s_j^l$. That is, $\overline{q_j^l}(0) = \overline{s_j^l} \neq s_j^l$. Now consider $q_j^*(0) = s_j^0 + rs_j^1 + \ldots + r^l\overline{s_j^l} + \ldots + r^\ell s_j^\ell$. We claim that with very high probability $q^*(j) \neq q_j^*(0)$. The probability that $q^*(j) = q_j^*(0)$ is same as the probability that two different degree-ℓ polynomials with coefficients $(s_j^0, \ldots, \overline{s_j^l}, \ldots, s_j^\ell)$ and $(s_j^0, \ldots, s_j^l, \ldots, s_j^\ell)$ respectively, intersect at a random value r. Since any two degree-ℓ polynomial can intersect each other at most at ℓ values, r has to be one of these ℓ values. But r is chosen randomly after the completion of all ACSS-MS-Share$_j$ for $j = 0, \ldots, n$ (so during executions of ACSS-MS-Share$_i$'s, D is unaware of r). So the above event can happen with probability at most $\frac{\ell}{|\mathbb{F}|} \approx \epsilon$. Thus with probability at least $1 - \epsilon$, $q^*(j) \neq q_j^*(0)$ and hence no honest party will terminate the protocol if D has t-$(1d)$-shared $\overline{S^j} \neq S^j$ for some j. So if some honest party has terminated t-(2d)-Share, then corrupted D must have attempted to t-$(1d)$-share correct S^i in each ACSS-MS-Share$_i$, for $i = 1, \ldots, n$. Now rest of the proof will follow from the proof of part 1 of **Correctness**.

Secrecy: When D is honest, ACSS-MS-Share$_0$ does not leak any information on $s^1, \ldots, s^\ell$ (from the secrecy property of ACSS-MS-Share). Later $s^* = q^*(0)$ does not leak any information about $s^1, \ldots, s^\ell$ during verification process, as s^0 is randomly chosen by D and therefore s^* will look completely random for the adversary $\mathcal{A}_t$. □

Lemma 10.17 *Protocol t-(2d)-Share privately communicates $\mathcal{O}((\ell n^5 + n^6 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-casts $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits.*

PROOF: Follows from the fact that in protocol t-(2d)-Share, there are $n + 1$ instances of ACSS-MS-Share, one instance of RNG, one instance of ACSS-MS-Rec-Public and n instances of ACSS-MS-Rec-Private. □

10.6 Preparation Phase

Here we generate t-$(1d)$-sharing of $c_M + c_R$ *secret* random *multiplication triples* (a^k, b^k, c^k), such that for $k = 1, \ldots, c_M + c_R$, $c^k = a^k b^k$. For this we first generate t-$(2d)$-sharing of *secret* random doubles $([[a^k]]_t, [[b^k]]_t)$ for $k = 1, \ldots, c_M + c_R$. Given these random doubles, we generate t-$(1d)$-sharing of c^k, for $k = 1, \ldots, c_M + c_R$, by adapting a technique from [48] which was given for synchronous settings.

10.6.1 Generating Secret Random t-$(2d)$-sharing

In section 10.5, we have presented a protocol called t-(2d)-Share which allows a $D \in \mathcal{P}$ to generate t-$(2d)$-sharing of ℓ secrets. We now present a protocol called Random-t-(2d)-Share which allows all the parties in $\mathcal{P}$ to jointly generate random t-$(2d)$-sharing of ℓ secrets, unknown to $\mathcal{A}_t$. The protocol terminates and generates its desired output except with probability ϵ. Random-t-(2d)-Share asks individual party to act as dealer and t-$(2d)$-share $\frac{\ell}{n-2t}$ random secrets. Then we run ACS protocol to agree on a common set of $n-t$ parties who have correctly t-$(2d)$-shared

$\frac{\ell}{n-2t}$ random secrets. Now out of these $n-t$ parties, at least $n-2t$ are honest. Hence the secrets that are t-$(2d)$-shared by these $n-2t$ honest parties are truly random and unknown to $\mathcal{A}_t$. So if we consider the $\frac{\ell}{n-2t}$ t-$(2d)$-sharing done by each of the honest parties in common set, then we will get $\frac{\ell}{n-2t} * (n-2t) = \ell$ random t-$(2d)$-sharing in total. For this, we use *Vandermonde Matrix* [52] and its ability to extract randomness which was also exploited in [141, 52, 13] (*Vandermonde Matrix* is recalled in Section 9.4.2 of Chapter 9). Protocol Random-t-(2d)-Share is now presented in Fig. 10.6.

To bound the error probability by ϵ, the computation of Random-t-(2d)-Share is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^7 2^{-\kappa} \max(4n, \kappa)$. This is derived from the fact that in Random-t-(2d)-Share, protocol t-(2d)-Share is invoked with $\frac{\epsilon}{n}$ error probability and as mentioned in previous section, $\epsilon \geq n^6 2^{-\kappa} \max(4n, \kappa)$ should hold to bound error probability of t-(2d)-Share by ϵ. So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

Lemma 10.18 *Protocol Random-t-(2d)-Share satisfies the following properties:*

1. **Termination**: *All honest parties eventually terminate the protocol with probability at least* $(1-\epsilon)$.

2. **Correctness**: *The protocol outputs correct* t-$(2d)$-*sharing of* ℓ *values with probability at least* $(1-\epsilon)$.

3. **Secrecy**: *The* ℓ *values whose* t-$(2d)$-*sharing is generated by the protocol will be completely random and unknown to* $\mathcal{A}_t$.

PROOF: **Termination:** By the **Termination** property of t-(2d)-Share (see Lemma 10.16), every instance of t-(2d)-Share initiated by an honest P_i as a dealer will be eventually terminated by all honest parties, except with probability ϵ'. Moreover, if an honest party terminates an instance of t-(2d)-Share (initiated by some party), then eventually every other honest party will terminate that instance of t-(2d)-Share , except with probability ϵ'. Now since there are at least $2t+1$ honest parties, except with probability $(2t+1)\epsilon' \approx \epsilon$, all instances of t-(2d)-Share initiated by honest parties will be terminated by every honest party. So eventually protocol ACS will output a common set C of size $n-t$, except with probability ϵ, such that all instances of t-(2d)-Share initiated by the parties in C will be eventually terminated by all honest parties in $\mathcal{P}$. This proves the **Termination** property.

Correctness: From the **Correctness** property of t-(2d)-Share (see Lemma 10.16), each instance of t-(2d)-Share initiated by a party in C will correctly generate t-$(2d)$-sharing of corresponding secrets, except with probability ϵ'. So with probability at most $(n-t)\epsilon' \approx \epsilon$, all instances of t-(2d)-Share initiated by the parties in C will correctly generate t-$(2d)$-sharing of corresponding secrets. Hence except with probability ϵ, protocol Random-t-(2d)-Share will correctly generate the t-$(2d)$-sharing of ℓ values. This proves the **Correctness** property.

Secrecy: From the **Secrecy** property of t-(2d)-Share (see Lemma 10.16), the values which are t-$2d$-shared by an honest party using t-(2d)-Share are completely random and are unknown to $\mathcal{A}_t$. Now there are at least $(n-t)-t = n-2t$ honest

Figure 10.6: Protocol for Collectively Generating t-($2d$)-sharing of ℓ secrets, $n = 3t+1$

Protocol Random-t-(2d)-Share($\mathcal{P}, \ell, \epsilon$)

Code for P_i: — Every party executes this code

1. Select $L = \frac{\ell}{n-2t}$ random secret elements $(s^{(i,1)}, \ldots, s^{(i,L)})$. As a dealer, invoke t-(2d)-Share($P_i, \mathcal{P}, S^i, \epsilon'$), with $\epsilon' = \frac{\epsilon}{n}$, to generate t-($2d$)-sharing of $S^i = (s^{(i,1)}, \ldots, s^{(i,L)})$.
2. For $j = 1, \ldots, n$, participate in t-(2d)-Share($P_j, \mathcal{P}, S^j, \epsilon'$).

Agreement on a Common Set: Code for P_i — Every party executes this code

1. Create a set $C^i = \emptyset$. Upon terminating t-(2d)-Share($P_j, \mathcal{P}, S^j, \epsilon'$), include P_j in C^i.
2. Take part in ACS with the set C^i as input.

Generation of Random t-($2d$)-sharing: Code for P_i: — Every party executes this code

1. Wait until ACS completes with output C containing $n-t$ parties. For every $P_j \in C$, obtain the i^{th} shares $s_i^{(j,1)}, \ldots, s_i^{(j,L)}$ of S^j and i^{th} share-shares $s_{ki}^{(j,1)}, \ldots, s_{ki}^{(j,L)}$ of shares $s_k^{(j,1)}, \ldots, s_k^{(j,L)}$, corresponding to each P_k, for $k = 1, \ldots, n$. Without loss of generality, let $C = \{P_1, \ldots, P_{n-t}\}$.
2. Let V denote an $(n-t) \times (n-2t)$ publicly known *Vandermonde Matrix*.
   (a) For every $k \in \{1, \ldots, L\}$, let $(r^{(1,k)}, \ldots, r^{(n-2t,k)}) = (s^{(1,k)}, \ldots, s^{(n-t,k)})V$.
   (b) Locally compute i^{th} share of $r^{(1,k)}, \ldots, r^{(n-2t,k)}$ as $(r_i^{(1,k)}, \ldots, r_i^{(n-2t,k)}) = (s_i^{(1,k)}, \ldots, s_i^{(n-t,k)})V$.
   (c) For each $1 \leq j \leq n$, locally compute the i^{th} share-share of share $(r_j^{(1,k)}, \ldots, r_j^{(n-2t,k)})$ as $(r_{ji}^{(1,k)}, \ldots, r_{ji}^{(n-2t,k)}) = (s_{ji}^{(1,k)}, \ldots, s_{ji}^{(n-t,k)})V$ and terminate Random-t-(2d)-Share.

The values $r^{(1,1)}, \ldots, r^{(n-2t,1)}, \ldots, r^{(1,L)}, \ldots, r^{(n-2t,L)}$ denote the ℓ random secrets which are t-($2d$)-shared.

parties in C and hence the L values which are t-$2d$-shared by each them will be completely random and unknown to $\mathcal{A}_t$. Now from the randomness extraction property of Vandermonde Matrix, the values $r^{(1,1)}, \ldots, r^{(n-2t,1)}, \ldots, r^{(1,L)}, \ldots, r^{(n-2t,L)}$ will be completely random and unknown to $\mathcal{A}_t$. This proves the **Secrecy** property. $\square$

Lemma 10.19 *Protocol Random-t-(2d)-Share privately communicates $\mathcal{O}((\ell n^5 + n^7 \log\frac{1}{\epsilon}) \log\frac{1}{\epsilon})$ bits and A-casts $\mathcal{O}(n^6 \log\frac{1}{\epsilon})$ bits.*

PROOF: The communication complexity follows from the fact that in protocol Random-t-(2d)-Share, n instances of t-(2d)-Share, each dealing with $\frac{\ell}{n-2t} = \frac{\ell}{\Theta(n)}$

values and having an error probability of $\frac{\epsilon}{n}$ are executed. This will require private communication of $\mathcal{O}((\ell n^5 + n^7 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits. Moreover, the protocol requires one invocation of ACS which incurs A-cast communication of $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits. □

10.6.2 An ABC Protocol– Proving $c = ab$

The ABC protocol has been considered in synchronous network in Chapter 5 and 6. Here we design the protocol for asynchronous network, but the core techniques are almost similar to the one used in synchronous network. So consider the following problem: let $D \in \mathcal{P}$ has t-$(1d)$-shared ℓ pair of values $(a^1, b^1), \ldots, (a^\ell, b^\ell)$. Now D wants to t-$(1d)$-share $c^1, \ldots, c^\ell$ where $c^l = a^l b^l$, for $l = 1, \ldots, \ell$. Moreover, during this process, D does not want to leak any *additional* information about a^l, b^l and c^l. We propose a protocol ProveCeqAB to achieve this task in asynchronous settings, following a technique proposed in [48] for synchronous settings. In fact the idea for synchronous settings (following the technique of [48]) has been presented in Section 6.7. Hence, we directly present the protocol in Fig. 10.7.

In protocol ProveCeqAB, if D is honest then a^l, b^l and c^l will remain information theoretically secure. If D is honest, then every honest party will eventually complete ProveCeqAB, and if some honest party has completed ProveCeqAB, then all the honest parties will eventually complete ProveCeqAB.

To bound the error probability by ϵ, the computation of ProveCeqAB is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^5 2^{-\kappa} \max(4n, \kappa)$. This is derived from the fact that in ProveCeqAB,

1. ACSS-MS-Share is invoked with $\frac{\epsilon}{3}$ error probability and as mentioned in Section 10.2.3, $\epsilon \geq n^5 \kappa 2^{-\kappa}$ should hold to bound error probability of ACSS-MS by ϵ;

2. RNG is invoked with ϵ error probability and this enforces $\epsilon \geq n^5 2^{-\kappa} \max(4n, \kappa)$.

So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

Lemma 10.20 *Protocol ProveCeqAB satisfies the following properties:*

1. **Termination:** *If D is honest then all honest parties will terminate the protocol, except with probability ϵ. If some honest party terminates the protocol, then eventually every other honest party will terminate the protocol, except with probability ϵ.*

2. **Correctness:** *If some honest party terminates the protocol, then except with probability ϵ, D has t-$(1d)$-shared $c^l = a^l b^l$, for $l = 1, \ldots, \ell$.*

3. **Secrecy:** *If D is honest then a^l, b^l, c^l will be information theoretically secure for all $l = 1, \ldots, \ell$.*

PROOF: **Termination**: When D is honest, all three instances of ACSS-MS-Share will terminate and correctly generate t-$(1d)$-sharing of $\mathcal{B}$, $\mathcal{C}$ and Λ with probability at least $(1 - 3\frac{\epsilon}{3}) = (1 - \epsilon)$. Consequently in verification steps, both the instances of ACSS-MS-Rec-Public will terminate and reconstruct proper values with probability at least $1 - \epsilon$. Now it follows that the condition specified in step 4 under **Verifying whether $c^l = a^l.b^l$:** of the protocol will pass with probability

Figure 10.7: Protocol for Generating t-$(1d)$-sharing of $[c^1]_t = [a^1]_t.[b^1]_t, \ldots, [c^\ell]_t = [a^\ell]_t.[b^\ell]_t$, $n = 3t+1$

Protocol ProveCeqAB($D, \mathcal{P}, [a^1]_t, \ldots, [a^\ell]_t, [b^1]_t, \ldots, [b^\ell]_t, \epsilon$)

Sharing by D:

1. **Code for D:**
   (a) Select ℓ non-zero random elements $\beta^1, \ldots, \beta^\ell$. For $l = 1, \ldots, \ell$, let $c^l = a^l b^l$ and $d^l = b^l \beta^l$. Let $\mathcal{B} = (\beta^1, \ldots, \beta^\ell)$, $\mathcal{C} = (c^1, \ldots, c^\ell)$ and $\Lambda = (d^1, \ldots, d^\ell)$.
   (b) Invoke ACSS-MS-Share($D, \mathcal{P}, \mathcal{B}, \frac{\epsilon}{3}$), ACSS-MS-Share($D, \mathcal{P}, \mathcal{C}, \frac{\epsilon}{3}$) and ACSS-MS-Share($D, \mathcal{P}, \Lambda, \frac{\epsilon}{3}$).
2. **Code for P_i:** Participate in the ACSS-MS-Share protocols initiated by D to obtain the i^{th} share $(\beta_i^1, \ldots, \beta_i^\ell)$, $(c_i^1, \ldots, c_i^\ell)$ and $(d_i^1, \ldots, d_i^\ell)$ of $\mathcal{B}, \mathcal{C}$ and Λ respectively.

Verifying whether $c^l = a^l.b^l$: Code for P_i — Every party executes this code

1. Once the three instances of ACSS-MS-Share initiated by D are terminated, participate in protocol RNG to jointly generate a random non-zero value r with error probability ϵ.
2. For $l = 1, \ldots, \ell$, locally compute $p_i^l = ra_i^l + \beta_i^l$, the i^{th} share $p^l = ra^l + \beta^l$. Participate in ACSS-MS-Rec-Public($D, \mathcal{P}, (p^1, \ldots, p^\ell), \epsilon$) to publicly reconstruct p^l for $l = 1, \ldots, \ell$.
3. Upon reconstruction of p^l's, locally compute $q_i^l = p^l b_i^l - d_i^l - rc_i^l$ for $l = 1, \ldots, \ell$, to get the i^{th} share of $q^l = p^l b^l - d^l - rc^l$. Participate in ACSS-MS-Rec-Public($D, \mathcal{P}, (q^1, \ldots, q^\ell), \epsilon$) to publicly reconstruct q^l for $l = 1, \ldots, \ell$.
4. Upon reconstruction of q^l's, locally check whether for $l = 1, \ldots, \ell$, $q^l \stackrel{?}{=} 0$. If yes then terminate ProveCeqAB.

at least $1 - \epsilon$. Hence when D is honest then the protocol will terminate with probability at least $1-\epsilon$. If some honest party has terminated the protocol, then $q^l = 0$ has been satisfied for all l. Every other honest party will also check the same and terminate the protocol.

Correctness: If some honest party terminates the protocol, then it implies that $q^l = 0$ for all $l = 1, \ldots, \ell$. Now notice that $q^l = p^l b^l - d^l - rc^l = (ra^l + \beta^l)b^l - b^l\beta^l - rc^l = ra^l b^l - rc^l + \beta^l b^l - d^l = r(a^l b^l - c^l) + \beta^l b^l - d^l$. Now if corrupted D shares $c^l \neq a^l b^l$ and $d^l \neq \beta b^l$, then $q^l = r(a^l b^l - c^l) + \beta^l b^l - d^l$ will be non-zero, except for only one value of r. But since r is randomly generated, the probability that r is that value is $\frac{1}{|\mathbb{F}|} < \epsilon$ which is negligibly small. Thus if some honest party terminates the protocol, then with probability $(1-\epsilon)$, D has t-$(1d)$-shared a^l, b^l and c^l satisfying $c^l = a^l b^l$ for all $l = 1, \ldots, \ell$.

Secrecy: We now prove the secrecy of a^l, b^l, c^l for all $l = 1, \ldots, \ell$ when D is honest. From the secrecy property of ACSS-MS-Share, a^l, b^l, c^l will remain secure after their t-$(1d)$-sharing. Now we will show that both p^l and q^l will not leak any information about a^l, b^l, c^l. Clearly $p^l = (ra^l + \beta^l)$ will look completely random to adversary $\mathcal{A}_t$ as β^l is randomly chosen. Furthermore $q^l = 0$ and hence q^l does not leak any information on a^l, b^l, c^l. Hence the lemma. $\Box$

Lemma 10.21 *Protocol ProveCeqAB privately communicates $\mathcal{O}((\ell n^4 + n^6 \log \frac{1}{\epsilon})$ $\log \frac{1}{\epsilon})$ bits and A-casts $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits.*

Proof: The proof follows from the following facts: ProveCeqAB invokes (a) $\Theta(1)$ instances of ACSS-MS-Share and ACSS-MS-Rec-Public (this will require private communication of $\mathcal{O}((\ell n^4 + n^5 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^4 \log n)$ bits) (b) And one instance of RNG (this requires private communication of $\mathcal{O}(n^6 (\log \frac{1}{\epsilon})^2)$ bits and A-cast of $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits). $\Box$

10.6.3 Generating Multiplication Triples: The Main Protocol For Preparation Phase

We now outline protocol PreparationPhase which generates t-$(1d)$-sharing of $c_M + c_R$ random multiplication triples. We explain the idea for a single triplet (a, b, c). First, Random-t-(2d)-Share is invoked to generate t-$(2d)$-sharing of (a, b) which results in P_i holding i^{th} share of a and b, namely a_i and b_i respectively. Now if each P_i locally computes $e_i = a_i b_i$, then this results in $2t$-$(1d)$-sharing of c. But we want each (honest) P_i to hold c_i, where $(c_1, \ldots, c_n)$ is the t-$(1d)$-sharing of c. For this we adapt a technique given in [93] for synchronous settings: Each P_i invokes ProveCeqAB to t-$(1d)$-share e_i. Now an instance of ACS will be executed to agree on a common set of $n - t = 2t + 1$ parties whose instances of ProveCeqAB has been terminated. For simplicity let this set contains $P_1, \ldots, P_{2t+1}$. Since $e_1, \ldots, e_{2t+1}$ are $2t + 1$ distinct points on a degree-$2t$ polynomial, say $C(x)$ where $C(0) = c$ (and $C(i) = e_i$ for $i = 1, \ldots, 2t + 1$), by Lagrange interpolation formula [46], c can be computed as $c = \sum_{i=1}^{2t+1} r_i e_i$ where $r_i = \prod_{j=1, j \neq i}^{2t+1} \frac{x-j}{i-j}$. The vector $(r_1, \ldots, r_{2t+1})$ is called recombination vector [46] and is known publicly. Now to get t-$(1d)$-sharing of c, P_j locally computes $c_j = \sum_{i=1}^{2t+1} r_i e_{ij}$ where e_{ij} is j^{th} share of e_i. By the properties of ProveCeqAB, each P_i in common set has indeed t-$(1d)$-shared $e_i = a_i b_i$ with very high probability. So by performing the above computation, correct t-$(1d)$-sharing of $c = ab$ will be generated with very high probability. Moreover, a, b and c will remain secure. Protocol PreparationPhase is now given in Fig. 10.8.

To bound the error probability by ϵ, the computation of PreparationPhase is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq 2n^7 2^{-\kappa} \max(4n, \kappa)$. This is derived from the fact that in PreparationPhase,

1. Random-t-(2d)-Share is invoked with $\frac{\epsilon}{2}$ error probability and as mentioned in Section 10.6.1, $\epsilon \geq n^7 2^{-\kappa} \max(4n, \kappa)$ should hold to bound error probability of Random-t-(2d)-Share by ϵ;

2. ProveCeqAB is invoked with $\frac{\epsilon}{n}$ error probability and as mentioned in Section 10.6.2, $\epsilon \geq n^6 2^{-\kappa} \max(4n, \kappa)$ should hold to bound error probability of ProveCeqAB by ϵ.

Figure 10.8: Protocol for Generating t-$(1d)$-sharing of $c_M + c_R$ secret random multiple triples

Protocol PreparationPhase($\mathcal{P}, \epsilon$)

Code for P_i: — Every party executes this code

1. Participate in two instances of Random-t-(2d)-Share($\mathcal{P}, c_M + c_R, \frac{\epsilon}{2}$) to generate t-$(2d)$-sharing of $a^1, \ldots, a^{c_M+c_R}$ and $b^1, \ldots, b^{c_M+c_R}$. Obtain the i^{th} shares $a_i^1, \ldots, a_i^{c_M+c_R}, b_i^1, \ldots, b_i^{c_M+c_R}$ and share-shares $a_{ji}^1, \ldots, a_{ji}^{c_M+c_R}, b_{ji}^1, \ldots, b_{ji}^{c_M+c_R}$, for $j = 1, \ldots, n$.
2. Let $c^k = a^k b^k$, for $k = 1, \ldots, c_M + c_R$. Upon termination of both the instances of Random-t-(2d)-Share, invoke ProveCeqAB($P_i, \mathcal{P}, [a_i^1]_t, \ldots, [a_i^{c_M+c_R}]_t, [b_i^1]_t, \ldots, [b_i^{c_M+c_R}]_t, \frac{\epsilon}{n}$) as a dealer, to generate t-$(1d)$-sharing of $c_i^1, \ldots, c_i^{c_M+c_R}$, where c_i^k is the i^{th} share of c^k. We refer the instance of ProveCeqAB initiated by P_i as ProveCeqAB$_i$.
3. For $j = 1, \ldots, n$, participate in ProveCeqAB$_j$.

Agreement on a Common Set: Code for P_i — Every party executes this code

1. Create a set $C^i = \emptyset$. Upon completing ProveCeqAB$_j$ initiated by P_j as a dealer, add P_j in C^i.
2. Take part in ACS with the set C^i as input.

Generation of t-$(1d)$-sharing of $c^1, \ldots, c^{c_M+c_R}$: Code for P_i — Every party executes this code

1. Wait until ACS completes with output C containing $2t+1$ parties. For simplicity, assume that $C = \{P_1, \ldots, P_{2t+1}\}$.
2. For $k = 1, \ldots, c_M + c_R$, locally compute $c_i^k = \sum_{j=1}^{2t+1} r_j c_{ji}^k$ the i^{th} share of $c^k = r_1 c_1^k + \ldots + r_{2t+1} c_{2t+1}^k$, where $(r_1, \ldots, r_{2t+1})$ is the publicly known recombination vector.

So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

We now prove the properties of protocol PreparationPhase.

Lemma 10.22 *Protocol PreparationPhase satisfies the following properties:*

1. **Termination**: *All honest parties will eventually terminate PreparationPhase, except with probability ϵ.*
2. **Correctness**: *The protocol correctly outputs t-$(1d)$-sharing of $c_M + c_R$ multiplication triples, except with probability ϵ.*
3. **Secrecy**: *The adversary $\mathcal{A}_t$ will have no information about (a^k, b^k, c^k), for $k = 1, \ldots, c_M + c_R$.*

PROOF: **Termination:** Following the termination property of Random-t-(2d)-Share, both the instances of Random-t-(2d)-Share will terminate except with probability $2\frac{\epsilon}{2} = \epsilon$. Now ProveCeqAB$_j$ invoked by an honest P_j will be eventually terminated by all honest parties, except with probability $\frac{\epsilon}{n}$. Moreover, if some honest party terminates protocol ProveCeqAB$_j$ for any P_j, then eventually every other honest party will terminate the protocol, except with probability $\frac{\epsilon}{n}$. Now since there are at least $2t+1$ honest parties, except with probability $(2t+1)\frac{\epsilon}{n} \approx \epsilon$, at least $2t+1$ instances of ProveCeqAB will be eventually terminated by all honest parties. So eventually, all honest parties will agree on a common set C containing $n-t$ parties, except with probability ϵ, such that the instances of ProveCeqAB initiated by every party in C is terminated by all honest parties in $\mathcal{P}$. Once this is done, every party will terminate protocol PreparationPhase after doing location computation. So all honest parties will terminate protocol PreparationPhase with probability at least $(1-\epsilon)$.

Correctness: Follows from the correctness property of protocol Random-t-(2d)-Share and ProveCeqAB.

Secrecy: Follows from the secrecy property of protocol Random-t-(2d)-Share and ProveCeqAB. □

Lemma 10.23 *Protocol PreparationPhase privately communicates $\mathcal{O}(((c_M+c_R)n^5+n^7\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits and A-casts $\mathcal{O}(n^7\log\frac{1}{\epsilon})$ bits.*

PROOF: Follows from the fact that the protocol invokes n instances of ProveCeqAB (this requires private communication of $\mathcal{O}(((c_M+c_R)n^5+n^7\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^7\log\frac{1}{\epsilon})$ bits) and two instances of Random-t-(2d)-Share (this requires private communication of $\mathcal{O}(((c_M+c_R)n^5+n^7\log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^6\log\frac{1}{\epsilon})$ bits) and one instance of ACS (this requires A-cast of $\mathcal{O}(n^6\log\frac{1}{\epsilon})$ bits). □

10.7 Input Phase

In protocol InputPhase, each P_i acts as a dealer to t-$(1d)$-share his input X_i containing c_i values. So $c_I = \sum_{i=1}^{n} c_i$, where c_I is the number of input gates in the circuit. The parties then agree on a set of at least $n-t$ parties (whose inputs will be taken into consideration for computation), by executing an ACS. Protocol InputPhase is now presented in Fig. 10.9.

To bound the error probability by ϵ, the computation of InputPhase is performed over a field $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq n^6\kappa 2^{-\kappa}$. This is derived from the fact that in InputPhase, ACSS-MS is invoked with $\frac{\epsilon}{n}$ error probability and as mentioned in Section 10.2.3, $\epsilon \geq n^5\kappa 2^{-\kappa}$ should hold to bound error probability of ACSS-MS by ϵ. So here each element from the field is represented by $\kappa = \log|\mathbb{F}| = \mathcal{O}(\log\frac{1}{\epsilon})$ bits.

Lemma 10.24 *Protocol InputPhase satisfies the following properties:*

1. **Termination**: *All honest parties will eventually terminate the protocol, except with probability ϵ.*

Figure 10.9: Protocol for Input Phase, $n = 3t + 1$

Protocol InputPhase($\mathcal{P}, \epsilon$)

Secret Sharing: Code for P_i — Every party executes this code

1. On input X_i, invoke ACSS-MS-Share($P_i, \mathcal{P}, X_i, \frac{\epsilon}{n}$) as a dealer to generate t-($1d$)-sharing of X_i.
2. For every $j = 1, \ldots, n$, participate in ACSS-MS-Share($P_j, \mathcal{P}, X_j, \frac{\epsilon}{n}$).

Agreement on a Common Set: Code for P_i — Every party executes this code

1. Create a set $C^i = \emptyset$. Upon completing ACSS-MS-Share($P_j, \mathcal{P}, X_j, \frac{\epsilon}{n}$) invoked by P_j as a dealer, add P_j in C^i.
2. Participate in ACS with the set C^i as input.
3. Output common set C containing $n - t$ parties and local shares of all inputs corresponding to parties in C.

2. **Correctness**: *The protocol correctly outputs t-($1d$)-sharing of inputs of the parties in agreed common set C, except with probability ϵ.*

3. **Secrecy**: *The adversary $\mathcal{A}_t$ will have no information about the inputs of the honest parties in set C.*

PROOF: The proof follows from the properties of ACSS-MS-Share. □

Lemma 10.25 *Protocol InputPhase privately communicates $\mathcal{O}((c_I n^4 + n^6 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-casts $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits.*

PROOF: Follows from the following facts: InputPhase invokes (a) n instances of ACSS-MS with $\ell = c_i$ for $i = 1, \ldots, n$ (this requires private communication of $\mathcal{O}((c_I n^4 + n^6 \log \frac{1}{\epsilon}) \log \frac{1}{\epsilon})$ bits and A-cast of $\mathcal{O}(n^4 \log n)$) (b) one instance of ACS (this requires A-cast communication of $\mathcal{O}(n^6 \log \frac{1}{\epsilon})$ bits). □

10.8 Computation Phase

Once the input phase is over, in the computation phase, the circuit is evaluated gate by gate, where all inputs and intermediate values are t-($1d$)-shared among the parties. As soon as a party holds his shares of the input values of a gate, he joins the computation of the gate.

Due to the linearity of t-($1d$)-sharing, linear gates can be computed locally by applying the linear function to the shares, i.e. for any linear function $c = f(a, b)$, the sharing $[c]_t$ is computed by letting every party P_i to compute $c_i = f(a_i, b_i)$, where a_i, b_i and c_i are the i^{th} shares of a, b and c respectively. With every random gate, one random triple (from the preparation phase) is associated, whose first component is directly used as outcome of the random gate. With every multiplication gate, one random triple (from the preparation phase) is associated, which is then used to compute t-$(1d)$-sharing of the product, following the circuit randomization technique of Beaver [5]. Given a pre-generated random multiplication triple (which is already correctly t-$(1d)$-shared) *Circuit Randomization* [5] allows to evaluate a multiplication gate at the cost of two public reconstructions. Let $z = xy$, where x, y are the inputs of the multiplication gate. Now z can be expressed as $z = ((x - a) + a)((y - b) + b) = (\alpha + a)(\beta + b)$, where (a, b, c) is a random multiplication triple. So given $([a]_t, [b]_t, [c]_t)$, $[z]_t$ can be computed as $[z]_t = \alpha\beta + \alpha[b]_t + \beta[a]_t + [c]_t$ after reconstructing α and β publicly. The security follows from the fact that α and β are random, for a random (a, b, c). Protocol ComputationPhase is now presented in Fig. 10.10.

To bound the error probability by ϵ, the computation of ComputationPhase is performed over the same field of PreparationPhase i.e $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq 2n^7 2^{-\kappa} \max(4n, \kappa)$ (this condition also accommodates the condition requires for InputPhase). So here each element from the field is represented by $\kappa = \log |\mathbb{F}| = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

Figure 10.10: Protocol for Computation Phase (Evaluating the Circuit), $n = 3t + 1$

Protocol ComputationPhase($\mathcal{P}, \epsilon$)

For every gate in the circuit: Code for P_i — Every party executes this code Wait until the i^{th} share of each of the inputs of the gate is available. Now depending on the type of the gate, proceed as follows:

1. **Input Gate:** $[s]_t = \mathsf{IGate}([s]_t)$: There is nothing to be done here.

2. **Linear Gate:** $[z]_t = \mathsf{LGate}([x]_t, [y]_t, \ldots)$: Compute $z_i = LGate(x_i, y_i, \ldots)$, the i^{th} share of $z = \mathsf{LGate}(x, y, \ldots)$, where $x_i, y_i, \ldots$ denotes i^{th} share of $x, y, \ldots$.

3. **Multiplication Gate:** $[z]_t = \mathsf{MGate}([x]_t, [y]_t, ([a^k]_t, [b^k]_t, [c^k]_t))$:

   (a) Let $([a^k]_t, [b^k]_t, [c^k]_t)$ be the random triple associated with the multiplication gate.

   (b) Compute $\alpha_i = x_i - a_i$ and $\beta_i = y_i - b_i$, the i^{th} share of $\alpha = (x - a)$ and $\beta = (y - b)$ respectively.

   (c) Participate in ACSS-Rec-Public to reconstruct α and β.

   (d) Upon reconstructing α and β, compute $z_i = \alpha\beta + \alpha b_i + \beta a_i + c_i$, the i^{th} share of $z = \alpha\beta + \alpha b + \beta a + c = xy$.

4. **Random Gate:** $[r]_t = \mathsf{RGate}([a^k]_t, [b^k]_t, [c^k]_t)$: Let $([a^k]_t, [b^k]_t, [c^k]_t)$ be the random triple associated with the random gate. Compute $r_i = a_i^k$ as the i^{th} share of r.

5. **Output Gate:** $x = \mathsf{OGate}([x]_t)$: Participate in ACSS-Rec-Public to reconstruct x.

Lemma 10.26 *Given that protocol* PreparationPhase *and* InputPhase *satisfy their* *properties specified in Lemma 10.22 and Lemma 10.24 respectively, Protocol* ComputationPhase *satisfies the following with probability at least* $(1-\epsilon)$*:*

1. **Termination***: All honest parties will eventually terminate the protocol.*
2. **Correctness***: Given* t*-*$(1d)$*-sharing of* $c_M + c_R$ *secret random triples, the protocol computes the outputs of the circuit correctly and privately.*

PROOF: Given that protocol PreparationPhase and InputPhase satisfy their **Termination** property specified in Lemma 10.22 and Lemma 10.24 respectively, termination of protocol ComputationPhase follows from the finiteness of the circuit representing function f and the termination property of ACSS-Rec-Public. Protocol PreparationPhase terminates with proper t-$(1d)$-sharing of c_M+c_R secret random triples, except with probability ϵ. Also protocol InputPhase terminates with proper t-$(1d)$-sharing of the inputs of the parties in common set C, except with probability ϵ. Hence protocol ComputationPhase will correctly compute the circuit and eventually terminate with probability at least $(1-\epsilon)$. □

Lemma 10.27 *Protocol* ComputationPhase *privately communicates* $O(n^2(c_M + c_O)\log\frac{1}{\epsilon})$ *bits.*

PROOF: Follows from the fact that in protocol ComputationPhase, $2c_M + c_O$ instances of ACSS-Rec-Public are executed, corresponding to c_M multiplication gates and c_O output gates. □

10.9 The New Statistical AMPC Protocol with Optimal Resilience

Now our new AMPC protocol called AMPC for evaluating function f which is represented by a circuit containing c_I, c_L, c_M, c_R and c_O input, linear, multiplication, random and output gates respectively, is as follows: (1). Invoke PreparationPhase$(\mathcal{P},\epsilon)$; (2). Invoke InputPhase$(\mathcal{P},\epsilon)$; (3). Invoke ComputationPhase$(\mathcal{P},\epsilon)$.

To bound the error probability by ϵ, the computation of AMPC is performed over the same field of PreparationPhase i.e. $\mathbb{F} = GF(2^\kappa)$, where κ has to be determined using the relation $\epsilon \geq 2n^7 2^{-\kappa}\max(4n,\kappa)$. So here each element from the field is represented by $\kappa = \log|\mathbb{F}| = \mathcal{O}(\log\frac{1}{\epsilon})$ bits.

Theorem 10.28 *Let* $n = 3t+1$*. Then protocol* AMPC *satisfies the following properties:*

1. **Termination***: Except with probability* ϵ*, all honest parties will eventually terminate the protocol.*
2. **Correctness***: Except with probability* ϵ*, the protocol correctly computes the outputs of the circuit.*
3. **Secrecy***: The adversary* $\mathcal{A}_t$ *will get no extra information other than what can be inferred by the input and output of the corrupted parties.*
4. **Communication Complexity***: The protocol privately communicates* $\mathcal{O}((c_I n^4 + c_M n^5 + c_R n^5 + c_O n^2 + n^7 \log\frac{1}{\epsilon})\log\frac{1}{\epsilon})$ *bits,* A-casts $\mathcal{O}(n^7\log\frac{1}{\epsilon})$ *bits.*

PROOF: The proof follows from the properties of protocol Preparation Phase, Input Phase and Computation Phase. □

10.10 Conclusion and Open Problems

In this chapter, we have presented a new ACSS scheme which is an essential building block of statistical AMPC protocol with optimal resilience (i.e., with $n = 3t + 1$). In fact, our ACSS scheme is the first ACSS scheme in the literature (in asynchronous settings) with $n = 3t+1$ (The perfect AVSS protocols of [35, 13] with $n = 4t + 1$ are in fact ACSS protocols). Our ACSS when employed for designing AMPC results in significant improvement over the only known statistical AMPC protocol of [21] (which does not employ any ACSS). The design approach of our ACSS are novel and first of their kind. We complete this chapter with the following interesting open problem:

Open Problem 17 *How to further reduce the communication complexity of our ACSS scheme which may lead to further reduction in the communication complexity of AMPC?*

Chapter 11

Efficient Statistical AVSS Protocol With Non-Optimal Resilience and Perfect AVSS With Optimal Resilience

Since AVSS serves as one of the main building blocks for AMPC and ABA, naturally it has got attention from researchers. In this chapter, we focus on AVSS and make a major contribution towards it. It is known that statistical AVSS is possible iff $n \geq 3t+1$ and on the other hand perfect AVSS is possible iff $n \geq 4t+1$. Thus a statistical AVSS with $n = 3t+1$ and likewise a perfect AVSS protocol with $n = 4t+1$ is said to have optimal resilience.

In this chapter, we design two AVSS protocols with $4t+1$ parties in which one is statistical (and thus have non-optimal resilience) and the other one is perfect along with being optimally resilient. Both our AVSS protocols are based on completely disjoint techniques. Yet, both our AVSSs achieve some interesting property that is never achieved before by any AVSS with $4t+1$ parties. In Chapter 10, we presented an ACSS scheme with $n = 3t+1$ that outputs t-($1d$)-sharing of secret(s). Our AVSSs in this chapter achieve beyond that and are capable of generating τ-($1d$)-sharing of secret(s) for any $t \leq \tau \leq 2t$ with much less communication complexity than the ACSS of Chapter 10. When we have $n = 4t+1$ parties, τ-($1d$)-sharing tremendously simplifies the computation of multiplication gate in an AMPC protocol. In the next chapter, the statistical AVSS and the perfect AVSS are respectively used for constructing our statistical AMPC and perfect AMPC with $n = 4t+1$. There we show how our AVSS protocols simplify computation of a multiplication gate.

A different interpretation of the newly achieved property of our AVSS protocols reveals that the amortized cost of sharing a single secret using our AVSSs is only $\mathcal{O}(n \log |\mathbb{F}|)$ bits. This is a clear improvement over the best known optimally resilient perfect AVSS of [13] whose amortized cost of sharing a single secret is $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits. In fact the AVSS of [13] was the best known AVSS among all protocols for AVSS with $n = 4t+1$ in terms of communication complexity.

Lastly, we emphasize that our AVSS protocols are of independent interest as AVSS has lot of other applications in ABA and many other distributed computing tasks apart from AMPC.

11.1 Introduction

11.1.1 The Network and Adversary Model

This is same as described in Section 8.1.1. Here we recall that the set of parties is denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$ and t out of the n parties can be under the influence of a *computationally unbounded Byzantine (active) adversary*, denoted as $\mathcal{A}_t$. We emphasize that we use $n = 4t + 1$ in this chapter.

11.1.2 The Definitions

For the current as well as next two chapters we will talk about AVSS that satisfies strong definition i.e even corrupted D is bound to commit secrets only from field $\mathbb{F}$. Hence here we override the definitions presented in Chapter 8 and present clean definition for perfect and statistical AVSS that will be valid for next two chapters as well.

Definition 11.1 (Perfect AVSS [19, 35]) *Let (*Sh*,*Rec*) be a pair of protocols in which a dealer $D \in \mathcal{P}$ shares a secret s from a finite field $\mathbb{F}$ using* Sh*. We say that (*Sh*,* Rec*) is a t-resilient AVSS scheme with n parties if the following hold for every possible $\mathcal{A}_t$:*

1. **Termination***:*
   (a) *If D is honest then each honest party will eventually terminate protocol* Sh*.*
   (b) *If some honest party has terminated protocol* Sh*, then irrespective of the behavior of D, each honest party will eventually terminate* Sh*.*
   (c) *If all the honest parties have terminated* Sh *and all the honest parties invoke protocol* Rec*, then each honest party will eventually terminate* Rec*.*
2. **Correctness***:*
   (a) *If D is honest then each honest party upon completing protocol* Rec*, outputs the shared secret s.*
   (b) *If D is faulty and some honest party has terminated* Sh*, then there exists a fixed $\overline{s} \in \mathbb{F}$, such that each honest party upon completing* Rec*, will output $\overline{s}$.*
3. **Secrecy***: If D is honest and no honest party has begun* Rec*, then $\mathcal{A}_t$ has no information about s.*

We now define statistical AVSS.

Definition 11.2 (Statistical AVSS) *This is same as the definition of Perfect AVSS (presented above), except that* **Termination** *and* **Correctness** *hold with probability $(1 - \epsilon)$ for some negligibly small error probability ϵ.*

The above definitions of AVSS can be extended for secret S containing multiple elements (say ℓ with $\ell > 1$) from $\mathbb{F}$.

Typically in applications like AMPC, sharing phase protocol of AVSSs are used to generate t-$(1d)$-sharing (for the definition of t-$(1d)$-sharing, see Definition 6.12) of secrets. Generalizing for t, we derive the following definition:

Definition 11.3 (**τ-(1d)-sharing**) *A value $s \in \mathbb{F}$ is said to be τ-(1d)-shared among the set of parties $\mathcal{P}$ if the following holds:*

- *there exists a random degree-τ polynomial $f(x)$ over $\mathbb{F}$, with $f(0) = s$ and*
- *each (honest) party $P_i \in \mathcal{P}$ holds $s_i = f(i)$, called i^{th} share of secret s.*

The vector of all $|\mathcal{P}|$ shares of s is called a τ-(1d)-sharing of s and is denoted by $[s]_\tau$.

Notice that a secret can be τ-(1d)-shared among any subset of $\mathcal{P}$, say $\overline{\mathcal{P}}$, provided that $|\overline{\mathcal{P}}| \geq \tau - t + 1$. The t-(1d)-sharing not only eases the computation of the circuit in an AMPC protocol, but the implementation of reconstruction phase of AVSS can be directly achieved using Online Error Correcting (OEC) technique proposed by [39, 35]. The sharing phase protocol of existing optimally resilient, perfect AVSS protocols [19, 13] directly outputs t-(1d)-sharing of secrets. On the other hand, some of the existing optimally resilient, statistical AVSS protocols were further used as black box to design some special protocols that generate t-(1d)-sharing of secrets (for example the ACSS in previous chapter). But so far there is no AVSS scheme for $n = 4t+1$ that is designed to generate τ-(1d)-sharing of secrets where $\tau > t$. This is exactly what our AVSS protocols (with $n = 4t+1$) achieve and thus using our AVSS we can generate $2t$-(1d)-sharing of secrets. The use of $2t$-(1d)-sharing in AMPC simplifies lot of computation for multiplication gate as shown in [13] (this will be discussed later in the next chapter).

11.1.3 Relevant Literature

1. *Perfect AVSS:* From [19, 35], perfect AVSS is possible iff $n \geq 4t + 1$. Hence, we call any *perfect* AVSS protocol with $n = 4t + 1$ as *optimally resilient, perfect* AVSS protocol. Such AVSS protocols are proposed in [19, 35, 13].

2. *Statistical AVSS:* Statistical AVSS is possible iff $n \geq 3t + 1$ [39, 21]. To the best of our knowledge, the AVSS protocols of [39, 21] and the protocols presented in this thesis in Chapter 8 are the only known *optimally resilient, statistical* AVSS protocols (i.e., with $n = 3t + 1$).

11.1.4 Contribution of This Chapter

In this chapter, we design two AVSS protocols with $4t + 1$ parties in which one is statistical and the other is perfect along with being optimally resilient. Both the AVSS protocols can generate τ-(1d)-sharing of secret for any τ, where $t \leq \tau \leq 2t$. They are the first AVSS of their kind in the literature as prior to our work, there is no AVSS that can generate τ-(1d)-sharing for $\tau > t$. Specifically, our AVSS protocols can generate τ-(1d)-sharing of $\ell \geq 1$ secrets from $\mathbb{F}$ concurrently, with a communication cost of $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits, where $\mathbb{F}$ is a finite field. In Table 11.1, we compare our AVSS with the existing AVSS protocols with $n = 4t + 1$ that generate t-(1d)-sharing of secrets.

Now it is to be noted that every τ-(1d)-sharing hides $(\tau + 1 - t)$ values that are completely unknown to the adversary. So a different interpretation leads us to say that our AVSS protocols share $\ell(\tau + 1 - t)$ secrets simultaneously with a communication cost of $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits. Putting $\tau = 2t$ (the maximum value of τ), we notice that the amortized cost of sharing a single secret using our AVSSs

Table 11.1: Comparison of our AVSS protocols with Existing AVSS Protocols. CC: Communication Complexity

Reference	Type	# Secrets	Sharing Generated	CC In Bits
[19]	Perfect	1	Only t-(1d)-sharing	$\mathcal{O}(n^3 \log(\|\mathbb{F}\|))$
[13]	Perfect	$\ell \geq 1$	Only t-(1d)-sharing	$\mathcal{O}(\ell n^2 \log(\|\mathbb{F}\|))$
This chapter	Statistical	$\ell \geq 1$	τ-(1d)-sharing, for any $t \leq \tau \leq 2t$	$\mathcal{O}(\ell n^2 \log(\|\mathbb{F}\|))$
This chapter	Perfect	$\ell \geq 1$	τ-(1d)-sharing, for any $t \leq \tau \leq 2t$	$\mathcal{O}(\ell n^2 \log(\|\mathbb{F}\|))$

is only $\mathcal{O}(n \log |\mathbb{F}|)$ bits. This is a clear improvement over the AVSS of [13] whose amortized cost of sharing a single secret is $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits.

Our statistical AVSS generates τ-(1d)-sharing by exploring several features of bivariate polynomial of different degrees in x and y and several other interesting observations.

To design our perfect AVSS, we exploit several interesting unexplored properties of (n,t)-star (a graph theoretic concept presented in section 4.4.2 of [35]) in conjunction with some properties of bivariate polynomial with *different* degree in variable x and y. The (n,t)-star was used to design an optimally resilient perfect AVSS protocol in [35] (the details of (n,t)-star are presented in Section 11.5 of current chapter). While the properties of (n,t)-star that our AVSS explores were not required in the AVSS of [35] (which generates only t-(1d)-sharing of secrets), our AVSS uses them for the first time for generating τ-(1d)-sharing of secrets, where $t \leq \tau \leq 2t$.

Our protocols for perfect AVSS work on a field $\mathbb{F}$ with $|\mathbb{F}| \geq n$. Hence every element from $\mathbb{F}$ can be represented by $\log |\mathbb{F}| = \mathcal{O}(\log n)$ bits. On the other hand, for statistical AVSS, we use two fields called ground field and extension field, which are defined as follows:

The Ground Field and The Extension Field: The field $\mathbb{F}$ that is used in perfect AVSS is denoted as *ground field*. Most of the computation of statistical AVSS is performed over this field. We also fix an extension field $\mathbb{E} \supset \mathbb{F}$ to be smallest extension for which $|\mathbb{E}| \geq 2^\kappa = \frac{1}{\epsilon}$, where ϵ is the error parameter. Each element of $\mathbb{E}$ can be represented using $\mathcal{O}(\kappa) = \mathcal{O}(\log \frac{1}{\epsilon})$ bits. We call $\mathbb{E}$ as *Extension Field*. Some of the computation of our statistical AVSS is performed over $\mathbb{E}$ so as to bound the error probability of the protocols by ϵ.

11.1.5 The Motivation for Presenting our AVSS Schemes

Our AVSS schemes share the following common properties:

1. Both the protocols are designed with $n = 4t + 1$;

2. Both the protocols have almost same communication complexity;

3. Both the schemes achieve τ-(1d)-sharing for any $t \leq \tau \leq 2t$.

However, one protocol is statistical (thus has non-optimal resilience) while the other one is perfect (thus has optimal resilience). Technique wise, both the AVSS schemes are completely independent. The main reason to present two different schemes is to show the difference in these techniques. We strongly believe that by suitable modifications to the techniques used in our statistical AVSS, we can further reduce its communication complexity while still maintaining its properties.

11.1.6 The Common Primitives Used for Both of our AVSS Schemes

Apart from A-cast that was recalled in Chapter 7, we need Online Error Correction (OEC) technique as the main tool for our AVSS protocols. We have discussed it in Section 10.2.1 of Chapter 10, specifically with respect to t-$(1d)$-sharing. Here we generalize and present it with respect to τ-$(1d)$-sharing.

11.1.6.1 Online Error Correction (OEC)

Let s be a secret which is τ-$(1d)$-shared among a set of parties $\overline{\mathcal{P}} \subseteq \mathcal{P}$, by a τ-degree polynomial $f(x)$, where $\tau < (|\overline{\mathcal{P}}| - 2t)$. Let $P_\alpha \in \mathcal{P}$ be a party, called as *receiver*, who wants to privately reconstruct s. This is done as follows: every party $P_i \in \overline{\mathcal{P}}$ sends his share s_i of s to P_α. The shares may reach P_α in any arbitrary order. Moreover, up to t of the shares may be incorrect or missing. To reconstruct $f(x)$, P_α applies OEC technique [19] on the received s_i's. Informally, OEC enables P_α to recognize when the received shares define a unique degree-τ interpolation polynomial. Specifically, P_α waits to receive $\tau + t + 1$ τ-consistent shares from the parties in $\overline{\mathcal{P}}$, such that these $\tau + t + 1$ shares lie on a unique τ-degree polynomial $f(x)$. Once P_α receives these values, he interpolates $f(x)$, outputs s and terminates. For details, see [19, 35].

Theorem 11.4 ([35, 13]) *Let $s \in \mathbb{F}$ be a secret, which is τ-$(1d)$-shared among a set of parties $\overline{\mathcal{P}} \subseteq \mathcal{P}$, where $\tau < (|\overline{\mathcal{P}}| - 2t)$. Then using OEC, any party $P_\alpha \in \mathcal{P}$ can privately reconstruct s. This requires private communication of $\mathcal{O}(n \log |\mathbb{F}|)$ bits.*

11.1.7 The Road-map

In Section 11.2 and 11.3, we present our statistical AVSS with $n = 4t + 1$ parties. For simplicity, we first present the protocol dealing with single secret in Section 11.2 and then extend it to the case of multiple secrets in Section 11.3. Section 11.4 states a different interpretation of our statistical AVSS sharing multiple secrets. Subsequently Section 11.5 recalls the notion of (n, t)-star, the algorithm for finding (n, t)-star and the properties of the algorithm. Next we present our perfect AVSS with $n = 4t + 1$ parties in Section 11.6 and 11.7. Again similar to the case of statistical AVSS, we present the protocol dealing with single secret in Section 11.6 and then extend it to the case of multiple secrets in Section 11.7. Section 11.8 states a different interpretation of our perfect AVSS sharing multiple secrets (parallel concept of what is stated for our statistical AVSS in Section 11.4). Finally this chapter ends with concluding remarks and open questions in Section 11.9.

11.2 Statistical AVSS For Sharing a Single Secret

We now present a novel, statistical AVSS scheme with $n = 4t + 1$, called St-AVSS consisting of a pair of protocols, (St-AVSS-Share,St-AVSS-Rec). Protocol St-AVSS-Share allows a dealer $D \in \mathcal{P}$ (dealer can be any party from $\mathcal{P}$) to generate τ-$(1d)$-sharing of a single secret from $\mathbb{F}$, where $t \leq \tau \leq 2t$. Protocol St-AVSS-Rec reconstructs the secret, given its τ-$(1d)$-sharing. Protocol St-AVSS has error probability of ϵ, meaning it satisfies **Termination** and **Correctness** of AVSS, except with error probability ϵ. Structurally, we divide protocol St-AVSS-Share into a sequence of following three phases:

1. **Distribution** Phase: Here D, on having a secret s, distributes information to the parties in $\mathcal{P}$.

2. **Verification & Agreement on CORE** Phase: Here the parties in $\mathcal{P}$ jointly perform some computation and communication in order to verify consistency of the information distributed by D in the previous phase. In case of successful verification, all the honest parties agree on a set of at least $3t+1$ parties, called $CORE$, satisfying certain properties (given in the sequel).

3. **Generation of τ-$(1d)$-sharing** Phase: In this phase, *only* the parties in $CORE$ communicate to every party in $\mathcal{P}$ and every party performs local computation (on the data received from the parties in $CORE$) to finally generate the τ-$(1d)$-sharing of secret s.

Each of the phases will be eventually completed by every honest party when D is honest. An honest party will terminate St-AVSS-Share, if it successfully completes the last phase, namely **Generation of τ-$(1d)$-sharing** Phase. If D is honest then each honest party will eventually terminate the last phase. Moreover, if D is corrupted and some honest party terminates the last phase, then each honest party will also eventually terminate the last phase (and hence St-AVSS-Share).

11.2.1 Distribution Phase

Here D on having a secret s, selects a random bivariate polynomial $F(x, y)$ over $\mathbb{F}$ of degree-(τ, t) (i.e degree τ in x and t in y), such that $F(0, 0) = s$. Let $f_i(x) = F(x, i)$ and $p_i(y) = F(i, y)$. While all $f_i(x)$ polynomials are of degree-τ, all $p_i(y)$ polynomials are of degree-t. We will call the $f_i(x)$ polynomials as *row polynomials* and $p_i(y)$ polynomials as *column polynomials*. Now D sends $f_i(x)$ to party P_i. In this phase, D also distributes some more information which will be used to keep his secret secure during **Verification & Agreement on CORE** phase. Precisely, D distributes the shares of $(t + 1)n$ random polynomials of degree-t which will be used for *blinding* purpose in **Verification & Agreement on CORE** phase. We refer these polynomials as *blinding polynomials*. The reason for taking $(t + 1)n$ blinding polynomials will be clear in the next section. The protocol for **Distribution** phase, called as St-Distr is given in Fig. 11.1.

Before proceeding further, we would like to mention a few interesting points about protocol St-Distr. The bivariate polynomial $F(x, y)$, selected by D, has degree-(τ, t). This results in each row polynomial to be of degree-τ and each column polynomial to be of degree-t. On the other hand, all the existing AVSS

Figure 11.1: First Phase of Protocol St-AVSS-Share: **Distribution** Phase

Protocol St-Distr($D, \mathcal{P}, s, \tau, \epsilon$)

Code for D: Only D executes this code

1. Select a random bivariate polynomial $F(x, y)$ of degree-(τ, t), such that $F(0, 0) = s$. For $i = 0, \ldots, n$, let $f_i(x) = F(x, i)$ and $p_i(y) = F(i, y)$.
2. Select $(t + 1)n$ degree-t, random, distinct *blinding polynomials*, over $\mathbb{F}$, denoted by $b^{(P_i,1)}(y), \ldots, b^{(P_i,t+1)}(y)$ for $i = 1, \ldots, n$.
3. For $i = 1, \ldots, n$, send the following to party P_i:
   (a) Row polynomial $f_i(x)$ of degree-τ;
   (b) For $j = 1, \ldots, n$, the shares $b^{(P_j,1)}(i), \ldots, b^{(P_j,t+1)}(i)$.

protocols (which generates only t-$(1d)$-sharing), based on the approach of bivariate polynomial, selects the degree of both x and y to be t [35, 13]. In subsequent phases, we create a situation, where the parties have to only reconstruct the column polynomials using the help of the parties in $CORE$, to complete the τ-$(1d)$-sharing. So even though the row polynomials may have degree more than t, the parties need not have to bother about reconstructing them.

We now describe **Verification & Agreement on CORE** phase. If **Verification & Agreement on CORE** phase is successful, then the secret s will be τ-$(1d)$-shared using degree-τ polynomial $f_0(x) = F(x, 0)$ in **Generation of τ-$(1d)$-sharing** phase.

11.2.2 Verification & Agreement on CORE Phase

As it is clear from the description of protocol St-Distr, if D behaves honestly then the *row polynomials* (i.e $f_i(x)$'s) held by the honest parties in $\mathcal{P}$, should define a unique degree-(τ, t) bivariate polynomial $F(x, y)$. But for a corrupted D, we must ensure that the above holds by enforcing some verification mechanism. While it may be difficult to ensure that the row polynomials of *all* the honest parties in $\mathcal{P}$ define a unique degree-(τ, t) bivariate polynomial (due to asynchrony of the network), it is easier to ensure the same for the honest parties in a set of at least $3t + 1$ parties, say $CORE$ ($CORE \subseteq \mathcal{P}$). In fact, this is what this phase attempts to achieve. The verification mechanism and the construction of $CORE$ is the crux of St-AVSS-Share. More clearly, $CORE$ has the following property:

Definition 11.5 (Properties of $CORE$) *$CORE(\subseteq \mathcal{P})$ is a set of at least $3t+1$ parties such that the row polynomials (received in* **Distribution** *Phase) of the honest parties in $CORE$ define a unique bivariate polynomial, say $\overline{F}(x, y)$, of degree-(τ, t). Moreover, if D is honest, then $\overline{F}(x, y) = F(x, y)$, where $F(x, y)$ was chosen by D in* **Distribution** *Phase.*

An Informal Description of the Current Phase: In our verification mechanism, every party has dual responsibility: (a) it acts as a verifier to verify consistency of the information distributed by D; (b) it also co-operates as a party, with other verifiers, in order to enable successful completion of the verification mechanism initiated by them. So, we first concentrate on the part of communication and computation that is to be carried out with respect to a single verifier, say V (here V can be any party from $\mathcal{P}$). *The goal of this part of communication and computation is to decide on a set of at least* $3t+1$ *parties, say* $AgreeSet^V$*, such that if* V *is honest, then* $AgreeSet^V$ *should satisfy all the desirable properties of* $CORE$*. In other words,* $AgreeSet^V$ *can be an eligible candidate for* $CORE$*, for an honest* V. To implement this, we design protocol Single-Verifier (given in Fig. 11.2). In the protocol if V is honest and some $AgreeSet^V$ is generated, then it is ensured that for $j = 1, \ldots, n$, the j^{th} point on row polynomials of all honest parties in $AgreeSet^V$ define some degree-t polynomial $\overline{p_j}(y)$ with high probability (Lemma 11.8). This further implies that the row polynomials held by the honest parties in $AgreeSet^V$ define a unique degree-(τ, t) bivariate polynomial with high probability (Lemma 11.9). We use the following notation in Single-Verifier:

Notation 11.6 *Given* ρ *polynomials,* $C = \{c_1(y), \ldots, c_\rho(y)\}$ *and a vector* $R = (\zeta_1, \ldots, \zeta_\rho)$ *of length* ρ*, we define* $c(y)$ *as the polynomial obtained by the linear combination of the polynomials in* C *with respect to* R*. That is,* $c(y) = \sum_{i=1}^{\rho} \zeta_i.c_i(y)$*. We express this by:* $c(y) = \mathsf{LCP}(C, R)$ *(LCP denotes Linear Combination of Polynomials). Similarly, we define* $c = \mathsf{LCV}(C, R)$ *(LCV denotes Linear Combination of Values), where* $C = \{c_1, \ldots, c_\rho\}$ *(set of* ρ *values) and* $c = \sum_{i=1}^{\rho} \zeta_i.c_i$

We now prove the following lemmas for Protocol Single-Verifier.

Lemma 11.7 *In protocol Single-Verifier, if* V *and* D *are honest, then eventually for some* $\beta \in \{1, \ldots, t+1\}$*,* $AgreeSet^{(V,\beta)}$ *with* $|AgreeSet^{(V,\beta)}| \geq 3t+1$ *will be generated.*

PROOF: For an honest D, every honest party in $\mathcal{P}$ will eventually send `Received-From-D` to V. Moreover, from protocol steps, V will A-cast distinct $ReceivedSet^{(V,\beta)}$ at most $t+1$ times. This is because $ReceivedSet^{(V,1)} \geq 3t+1$ and $n = 4t+1$. These facts together imply that eventually there will be a $\beta \in \{1, \ldots, t+1\}$, such that $ReceivedSet^{(V,\beta)}$ will contain all $3t+1$ honest parties. Moreover, each honest party from such $ReceivedSet^{(V,\beta)}$ will eventually enter into $AgreeSet^{(V,\beta)}$ when D is honest. So eventually $|AgreeSet^{(V,\beta)}|$ will be at least $3t+1$ for some β. □

Lemma 11.8 *In protocol Single-Verifier, if* V *is honest and some* $AgreeSet^{(V,\beta)}$ *(containing at least* $3t+1$ *parties) has been generated, then the following holds with probability at least* $(1-\epsilon)$*:*

1. *For all* $j = 1, \ldots, n$*, the* j^{th} *point on the row polynomials* $\overline{f_i}(x)$*s, held by the honest parties in* $AgreeSet^{(V,\beta)}$*, define degree-t polynomial* $\overline{p_j}(y)$*.*

2. *Moreover, the points on blinding polynomial* $b^{(V,\beta)}(y)$ *held by the honest parties in* $AgreeSet^{(V,\beta)}$ *will also lie on a degree-t polynomial.*

PROOF: If D is honest, then the lemma will be true, without any error. Hence we consider the case when D is corrupted. Let $H^{(V,\beta)}$ denote the set of honest parties in $AgreeSet^{(V,\beta)}$. First of all, since V is honest, he A-casts random $r^{(V,\beta)}$ only after listening `Received-From-D` signal from all the parties in $ReceivedSet^{(V,\beta)}$.

Figure 11.2: Steps to be executed with respect to a Single Verifier

Protocol Single-Verifier($V, \mathcal{P}, s, \tau, \epsilon$**)**

i. Code for P_i: Every party in $\mathcal{P}$, including D and V, executes this code.

1. Wait to receive $\overline{f_i}(x)$ and $b^{(P_j,1)}(i), \ldots, b^{(P_j,t+1)}(i)$ for $j = 1, \ldots, n$ from D.
2. After receiving, check whether $\overline{f_i}(x)$ is a degree-τ polynomial. If yes, then privately send a `Received-From-D` signal to V.

ii. Code for V: Only V executes this code.

1. Wait to obtain `Received-From-D` signal from $3t+1$ parties. Put the identities of these $3t+1$ parties in a set $ReceivedSet^{(V,1)}$. Select a random value $r^{(V,1)}$ from extension field $\mathbb{E}$ and A-cast $(r^{(V,1)}, ReceivedSet^{(V,1)})$.
2. After the previous step, for every new receipt of `Received-From-D` signal from $P_\alpha \notin ReceivedSet^{(V,\beta-1)}$ such that $1 < \beta \leq t+1$, construct $ReceivedSet^{(V,\beta)} = ReceivedSet^{(V,\beta-1)} \cup \{P_\alpha\}$, select a random $r^{(V,\beta)} \in \mathbb{E} \setminus \{r^{(V,1)}, \ldots, r^{(V,\beta-1)}\}$ and A-cast $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$.

iii. Code for D: Only D executes this code.

1. If $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$ is received from the A-cast of V, then A-cast the polynomial $E^{(V,\beta)}(y)$, where $E^{(V,\beta)}(y) = \mathsf{LCP}(\mathcal{E}, R)$. Here $\mathcal{E} = \{b^{(V,\beta)}(y), p_1(y), \ldots, p_n(y)\}$ and $R = (1, r^{(V,\beta)}, \left(r^{(V,\beta)}\right)^2, \ldots, \left(r^{(V,\beta)}\right)^n)$.

iv. Code for P_i: Every party in $\mathcal{P}$ executes this code.

1. If $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$ is received from the A-cast of V, then check if $P_i \in ReceivedSet^{(V,\beta)}$. If yes, then A-cast $e_i^{(V,\beta)} = \mathsf{LCV}(\Delta_i, R)$, where $\Delta_i = \{b^{(V,\beta)}(i), \overline{f_i}(1), \ldots, \overline{f_i}(n)\}$ and $R = (1, r^{(V,\beta)}, \left(r^{(V,\beta)}\right)^2, \ldots, \left(r^{(V,\beta)}\right)^n)$.
2. Say that **party P_j agrees with D with respect to** $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$, where $\beta \in \{1, \ldots, t+1\}$, if all the following hold:
   (a) $E^{(V,\beta)}(y)$ A-casted by D is a degree-t polynomial,
   (b) $P_j \in ReceivedSet^{(V,\beta)}$ and
   (c) $e_j^{(V,\beta)} = E^{(V,\beta)}(j)$, where $e_j^{(V,\beta)}$, $E^{(V,\beta)}(y)$ and $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$ are received from the A-casts of P_j, D and V respectively.
3. With respect to $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$, when there are $3t+1$ P_j's who agree with D, add them in a set $AgreeSet^{(V,\beta)}$.

Thus D has no knowledge of $r^{(V,\beta)}$, when he distributes the row polynomials and points on blinding polynomial $b^{(V,\beta)}(y)$ to the (honest) parties in $ReceivedSet^{(V,\beta)}$. Let $\overline{b^{(V,\beta)}}(y)$ denote the *minimum degree* polynomial, defined by the points on $b^{(V,\beta)}(y)$, held by the parties in $H^{(V,\beta)}$. Similarly, let $\overline{p_1}(y),\ldots,\overline{p_n}(y)$ denote the *minimum degree* column polynomials, defined by the points on the row polynomials, held by the parties in $H^{(V,\beta)}$. For convenience, we use a uniform notation for these $n+1$ polynomials. We denote them by $h^0(y),\ldots,h^n(y)$, respectively. Then the value $e_i^{(V,\beta)}$, A-casted by $P_i \in H^{(V,\beta)}$ is defined as $e_i^{(V,\beta)} = \sum_{j=0}^{n} \left(r^{(V,\beta)}\right)^j h^j(i)$.

We now claim that with probability at least $(1-\epsilon)$, all of $h^0(y),\ldots,h^n(y)$ have degree-t. On the contrary, if we assume that at least one of these polynomials has degree more than t, then we can show that the minimum degree polynomial, say $h^{min}(y)$, defined by $e_i^{(V,\beta)}$'s for $P_i \in H^{(V,\beta)}$ will be of degree more than t, with probability at least $(1-\epsilon)$. This will clearly imply $E^{(V,\beta)}(y) \neq h^{min}(y)$ and hence $e_i^{(V,\beta)} \neq E^{(V,\beta)}(i)$ for at least one $P_i \in H^{(V,\beta)}$. This is a contradiction as $e_i^{(V,\beta)} = E^{(V,\beta)}(i)$ holds for every $P_i \in Agree^{(V,\beta)}$ and $H^{(V,\beta)} \subseteq Agree^{(V,\beta)}$. This shows that our claim is true.

So we proceed to prove that $h^{min}(y)$ will be of degree more than t with probability at least $(1-\epsilon)$, when one of $h^0(y),\ldots,h^n(y)$ has degree more than t. For this, we show the following:

1. We first show that $h^{def}(y) = \Sigma_{j=0}^{n} \left(r^{(V,\beta)}\right)^j h^j(y)$ will be of degree more than t with probability at least $(1-\epsilon)$, if one of $h^0(y),\ldots,h^n(y)$ has degree more than t.

2. We then show that $h^{min}(y) = h^{def}(y)$, implying that $h^{min}(y)$ will be of degree more than t.

To prove the first point, assume that one of $h^0(y),\ldots,h^n(y)$, has degree more than t. Let m be such that $h^m(y)$ has maximal degree among $h^0(y),\ldots,h^n(y)$, and let t_m be the degree of $h^m(y)$. Then according to the condition, $t_m > t$. Note that $t_m < |H^{(V,\beta)}|$. This is because given $|H^{(V,\beta)}|$ values (recall that $h^0(y),\ldots,h^n(y)$ are defined by the points on the row polynomials, held by the honest parties in $H^{(V,\beta)}$), the maximum degree polynomial that can be defined using them is $|H^{(V,\beta)}|-1$. Now each $h^i(y)$ can be written as $h^i(y) = c_{t_m}^i y^{t_m} + \widehat{h^i}(y)$ where $\widehat{h^i}(y)$ has degree lower than t_m. Thus $h^{def}(y)$ can be written as:

$$\begin{aligned}
h^{def}(y) &= [c_{t_m}^0 y^{t_m} + \widehat{h^0}(y)] + r^{(V,\beta)}[c_{t_m}^1 y^{t_m} + \widehat{h^1}(y)] + \ldots + \left(r^{(V,\beta)}\right)^n [c_{t_m}^n y^{t_m} + \widehat{h^n}(y)] \\
&= y^{t_m}(c_{t_m}^0 + \ldots + \left(r^{(V,\beta)}\right)^n c_{t_m}^n) + \Sigma_{j=0}^{n} \left(r^{(V,\beta)}\right)^j \widehat{h^j}(y) \\
&= y^{t_m} c_{t_m} + \Sigma_{j=0}^{n} \left(r^{(V,\beta)}\right)^j \widehat{h^j}(y) \quad \text{where } c_{t_m} = c_{t_m}^0 + \ldots + \left(r^{(V,\beta)}\right)^n c_{t_m}^n
\end{aligned}$$

By assumption $c_{t_m}^m \neq 0$. It implies that the vector $(c_{t_m}^0,\ldots,c_{t_m}^n)$ is not a complete 0 vector. Hence $c_{t_m} = c_{t_m}^0 + \ldots + \left(r^{(V,\beta)}\right)^n c_{t_m}^n$ will be zero with probability $\frac{n}{|\mathbb{E}|-(\beta-1)} \approx 2^{-\Omega(\kappa)} \approx \epsilon$ (which is negligible), where $\beta \leq t+1$. This is because the vector $(c_{t_m}^0,\ldots,c_{t_m}^n)$ may be considered as the set of coefficients of a n degree polynomial, say $\mu(x)$, and hence the value c_{t_m} is the value of $\mu(x)$ evaluated at $r^{(V,\beta)}$. Now c_{t_m} will be zero if $r^{(V,\beta)}$ happens to be one of the n roots of $\mu(x)$ (since degree of $\mu(x)$ is at most n). Now since $r^{(V,\beta)}$ is chosen randomly from $\mathbb{E} \setminus \{r^{(V,1)},\ldots,r^{(V,\beta-1)}\}$ by V, independent of the polynomials $h^0(y),\ldots,h^n(y)$, the probability that it is a root of $\mu(x)$ is $\frac{n}{|\mathbb{E}|-(\beta-1)} \approx 2^{-\Omega(\kappa)} \approx \epsilon$. So with very high probability of $(1-\epsilon)$, c_{t_m} that is the t_m^{th} coefficient of $h^{def}(y)$ is non-zero. This implies that $h^{def}(y)$ will be of degree at least $t_m > t$. Notice that each $e_i^{(V,\beta)}$ (A-casted by P_i), corresponding to every $P_i \in H^{(V,\beta)}$ will lie on $h^{def}(y)$.

Now we show that $h^{min}(y) = h^{def}(y)$ and thus $h^{min}(y)$ has degree at least $t_m > t$. So consider the difference polynomial $dp(y) = h^{def}(y) - h^{min}(y)$. Clearly, $dp(y) = 0$, for all $y = i$, where $P_i \in H^{(V,\beta)}$. Thus $dp(y)$ will have at least $|H^{(V,\beta)}|$ roots. On the other hand, maximum degree of $dp(y)$ could be t_m, which is at most $|H^{(V,\beta)}| - 1$. These two facts together imply that $dp(y)$ is the zero polynomial, implying that $h^{def}(y) = h^{min}(y)$ and thus $h^{min}(y)$ has degree $t_m > t$. □

Lemma 11.9 *In protocol Single-Verifier, if V is honest and $AgreeSet^{(V,\beta)}$ (containing at least $3t+1$ parties) has been generated, then there exists a unique degree-(τ, t) bivariate polynomial $\overline{F}(x,y)$ such that row polynomial $\overline{f_i}(x)$ held by every honest $P_i \in AgreeSet^{(V,\beta)}$ satisfies $\overline{F}(x,i) = \overline{f_i}(x)$ with probability at least $(1-\epsilon)$. Moreover, if D is honest then $\overline{F}(x,y) = F(x,y)$.*

PROOF: Let l be the number of honest parties in $AgreeSet^{(V,\beta)}$. As $|AgreeSet^{(V,\beta)}| \geq 3t+1$, we have $l \geq 2t+1$. Without loss of generality, we assume that $P_1, \ldots, P_l$ are the honest parties in $AgreeSet^{(V,\beta)}$, holding the row polynomials $\overline{f_1}(x), \ldots, \overline{f_l}(x)$ respectively. Since V is honest and some $AgreeSet^{(V,\beta)}$ (containing at least $3t+1$ parties) has been generated, then from Lemma 11.8 with probability at least $(1-\epsilon)$, there are n degree-t column polynomials $\overline{p_1}(y), \ldots, \overline{p_n}(y)$ such that for every (i,j) with $i \in \{1, \ldots, l\}$ and $j \in \{1, \ldots, n\}$, we have $\overline{f_i}(j) = \overline{p_j}(i)$. We now claim that if this is the case, then there exists a unique bivariate polynomial $\overline{F}(x,y)$ of degree-(τ, t) over $\mathbb{F}$, such that for $i = 1, \ldots, l$, we have $\overline{F}(x,i) = \overline{f_i}(x)$ and for $j = 1, \ldots, n$, we have $\overline{F}(j,y) = \overline{p_j}(y)$. The proof is very similar to the proof of Lemma 4.26 of [35].

Let $V^{(k)}$ denote $k \times k$ Vandermonde matrix, where i^{th} column is $[i^0, \ldots, i^{k-1}]^T$, for $i = 1, \ldots, k$. Now consider the row polynomials $\overline{f_1}(x), \ldots, \overline{f_{t+1}}(x)$ and let E be the $(t+1) \times (\tau+1)$ matrix, where E_{ij} is the coefficient of x^j in $\overline{f_i}(x)$, for $i = 1, \ldots, t+1$ and $j = 0, \ldots, \tau$. Thus for $i = 1, \ldots, t+1$ and $j = 1, \ldots, \tau+1$, the $(i,j)^{th}$ entry in $E \cdot V^{(\tau+1)}$ is $\overline{f_i}(j)$.

Let $H = ((V^{(t+1)})^T)^{-1} \cdot E$ be a $(t+1) \times (\tau+1)$ matrix. Let for $i = 0, \ldots, \tau$, the $(i+1)^{th}$ column of H be $[r_{i0}, r_{i1}, \ldots, r_{it}]^T$. Now we define a degree-(τ, t) bivariate polynomial $\overline{F}(x,y) = \sum_{i=0}^{i=\tau} \sum_{j=0}^{j=t} r_{ij} x^i y^j$. Then from properties of bivariate polynomial, for $i = 1, \ldots, t+1$ and $j = 1, \ldots, \tau+1$, we have

$$\overline{F}(j,i) = (V^{(t+1)})^T \cdot H \cdot V^{(\tau+1)} = E \cdot V^{(\tau+1)} = \overline{f_i}(j) = \overline{p_j}(i)$$

This implies that for $i = 1, \ldots, t+1$, the polynomials $\overline{F}(x,i)$ and $\overline{f_i}(x)$ have same value at $\tau+1$ values of x. But since degree of $\overline{F}(x,i)$ and $\overline{f_i}(x)$ is τ, this implies that $\overline{F}(x,i) = \overline{f_i}(x)$. Similarly, for $j = 1, \ldots, \tau+1$, we have $\overline{F}(j,y) = \overline{p_j}(y)$.

Next, we will show that for any $t+1 < i \leq l$, the polynomial $\overline{f_i}(x)$ also lies on $\overline{F}(x,y)$. In other words, $\overline{F}(x,i) = \overline{f_i}(x)$, for $t+1 < i \leq l$. This is easy to show because according to theorem statement, $\overline{f_i}(j) = \overline{p_j}(i)$, for $j = 1, \ldots, \tau+1$ and $\overline{p_1}(i), \ldots, \overline{p_{\tau+1}}(i)$ lie on $\overline{F}(x,i)$ and uniquely defines $\overline{F}(x,i)$. Since both $\overline{f_i}(x)$ and $\overline{F}(x,i)$ are of degree τ, this implies that $\overline{F}(x,i) = \overline{f_i}(x)$, for $t+1 < i \leq l$. Similarly, we can show that $\overline{F}(j,y) = \overline{p_j}(y)$, for $\tau+1 < j \leq n$.

It is easy to see that if D is honest, then $\overline{f_i}(x) = f_i(x)$ and $\overline{p_i}(y) = p_i(y)$ and therefore $\overline{F}(x,y) = F(x,y)$. □

Lemma 11.10 *If D is honest then s will be secure in protocol Single-Verifier.*

PROOF: Without loss of generality, let $\mathcal{A}_t$ controls $P_1, \ldots, P_t$. So $\mathcal{A}_t$ will know $f_1(x), \ldots, f_t(x)$ and hence t points on $p_1(y), \ldots, p_n(y)$. $\mathcal{A}_t$ will also learn $E^{(V,\beta)}(y)$ for $\beta = 1, \ldots, t+1$ and t points on $b^{(V,1)}(y), \ldots, b^{(V,t+1)}(y)$. But each $E^{(V,\beta)}(y)$ is linear combination of $b^{(V,\beta)}(y), p_1(y), \ldots, p_n(y)$. As $b^{(V,\beta)}(y)$ is completely random and independent of $p_1(y), \ldots, p_n(y)$, $E^{(V,\beta)}(y)$ will be completely random for $\mathcal{A}_t$. Moreover for every $\beta \in \{1, \ldots, t+1\}$, distinct $b^{(V,\beta)}(y)$ is used. The rest now follows from the properties of degree-(τ, t) bivariate polynomial. □

So far, we concentrated on the communication that is to be carried out with respect to a single V. We proved that if V is honest then Single-Verifier can provide with a candidate solution for $CORE$ (see Lemma 11.7-11.9). But as we do not know the exact identities of honest parties, we can not pick an $AgreeSet^{(V,*)}$ for an honest V and assign that as $CORE$. Thus $CORE$ construction requires a more involved trick. Informally, we execute Single-Verifier for every $V \in \mathcal{P}$ and compute $CORE$ based on $AgreeSet^{(*,*)}$'s by exploring several important observations.

Remark 11.11 (Need for $n(t+1)$ Blinding Polynomials in St-Distr) *Recall that in protocol St-Distr, D has selected $n(t+1)$ blinding polynomials. The reason for this is as follows: From the proof of Lemma 11.7, a single verifier V will A-cast at most $t+1$ $ReceivedSet^{(V,\beta)}$'s. Now, from Lemma 11.10, in order to maintain the secrecy of s, distinct $b^{(V,\beta)}(y)$ should be used for computing $E^{(V,\beta)}(y)$ for every $\beta \in \{1, \ldots, t+1\}$. Now in* **Verification and Agreement on** *$CORE$ phase, each of the n parties will act as a verifier and execute protocol Single-Verifier. Hence D should select $n(t+1)$ blinding polynomials.*

Before presenting our protocol for **Verification & Agreement on CORE** phase, we capture several important observations in terms of the following lemmas which will help to grasp the part of code used for constructing $CORE$.

Lemma 11.12 *For an* **honest** *V, the row polynomials held by* **honest parties** *in $AgreeSet^{(V,\beta)}$ and $AgreeSet^{(V,\gamma)}$ with $\beta \neq \gamma$, define same degree-(τ, t) bivariate polynomial.*

PROOF: By Lemma 11.9, for an honest V, the row polynomials held by honest parties in $AgreeSet^{(V,\beta)}$ as well as in $AgreeSet^{(V,\gamma)}$ define unique degree-(τ, t) bivariate polynomials, say $\overline{F}(x, y)$ and $\widehat{F}(x, y)$ respectively with probability at least $(1 - \epsilon)$. Now $\overline{F}(x, y) = \widehat{F}(x, y)$, as they have at least $t+1$ row polynomials in common corresponding to at least $t+1$ *common honest parties* in $AgreeSet^{(V,\beta)}$ and $AgreeSet^{(V,\gamma)}$, who define a unique bivariate polynomial of degree-(τ, t). □

Lemma 11.13 *For any two* **honest** *verifiers V_α, V_δ, the row polynomials of honest parties in any $AgreeSet^{(V_\alpha,\beta)}, AgreeSet^{(V_\delta,\gamma)}$ with $\beta, \gamma \in \{1, \ldots, t+1\}$ define same degree-(τ, t) bivariate polynomial.*

PROOF: The proof for this lemma is almost same as lemma 11.12 and therefore follows from Lemma 11.9 and the fact that there are at least $t+1$ common honest parties in $AgreeSet^{(V,\beta)}$ and $AgreeSet^{(V,\gamma)}$, whose row polynomials define the same bivariate polynomial. □

Now the protocol for **Verification & Agreement on CORE** Phase is given in Fig. 11.3.

We now prove the properties of protocol St-Ver-Agree.

Figure 11.3: Second Phase of Protocol St-AVSS-Share: **Verification & Agreement on CORE** phase

Protocol St-Ver-Agree($D, \mathcal{P}, s, \tau, \epsilon$)

Verification and $CORE$ Construction:

i. **Code for P_i:** This code is executed by every party including D

1. Execute Single-Verifier($P_\alpha, \mathcal{P}, s, \tau, \epsilon$) for every verifier $P_\alpha \in \mathcal{P}$ in parallel.
2. Add a verifier P_α to a set $ValidVerifier$ if at least one $AgreeSet^{(P_\alpha,\beta)}$ for some $\beta \in \{1, \ldots, t+1\}$ has been generated.
3. Check whether $|ValidVerifier| \geq t+1$ and in case of 'yes' perform the following computation:
   (a) For every $P_\alpha \in ValidVerifier$, compute $AgreeSet^{P_\alpha} = \cup_\beta AgreeSet^{(P_\alpha,\beta)}$.
   (b) Compute $CORE_i = \{P_j \mid P_j$ belongs to $AgreeSet^{P_\alpha}$ for at least t+1 $P'_\alpha s$ in $ValidVerifier\}$.
   (c) Wait for new updates (such as generation of new set $AgreeSet^{(P_\alpha,\beta)}$, expansion of existing $AgreeSet^{(P_\alpha,\beta)}$'s etc.) and repeat the same computation (i.e steps 2-3((a),(b))) to update $CORE_i$ for every new update.

ii. **Code for D:** This code is executed only by D

1. A-cast $CORE = CORE_D$ as soon as $|CORE_D| = 3t+1$.

Agreement on CORE: Code for P_i:

1. Wait to receive $CORE$ from the A-cast of D such that $|CORE| = 3t+1$.
2. Wait until $CORE \subseteq CORE_i$ and then accept $CORE$.

Lemma 11.14 *The row polynomials held by the honest parties in $CORE$ define a unique degree-(τ, t) bivariate polynomial, say $\overline{F}(x, y)$ with probability at least $(1-\epsilon)$. Moreover, when D is honest then $\overline{F}(x, y) = F(x, y)$.*

Proof: By the construction of $CORE$, every party in $CORE$ is guaranteed to be present in $AgreeSet$ of at least one honest verifier. By Lemma 11.12, corresponding to an honest verifier P_α, the row polynomials held by the honest parties in $AgreeSet^{P_\alpha}$ define a unique degree-(τ, t) bivariate polynomial, say $\overline{F}(x, y)$, with probability at least $(1-\epsilon)$. Moreover, by Lemma 11.13, the row polynomials held by the honest parties in the union of $AgreeSet^{P_\alpha}$'s, corresponding to all *honest* P_α's, also define $\overline{F}(x, y)$ with probability at least $(1-\epsilon)$. This implies that the row polynomials held by the honest parties in $CORE$, define $\overline{F}(x, y)$ with probability at least $(1-\epsilon)$.

It is easy to see that for an honest D, $\overline{F}(x, y) = F(x, y)$ where $F(x, y)$ was the polynomial chosen by D in St-Distr. Moreover for an honest D, the row polynomials held by the honest parties in $CORE$ define $F(x, y)$ without any error. □

Once the parties agree on a $CORE$ set, generation of τ-$(1d)$-sharing requires n private reconstructions using OEC technique. We do that in **Generation of τ-$(1d)$-sharing** phase which is discussed in the sequel.

11.2.3 Generation of τ-$(1d)$-sharing Phase

Assuming that the honest parties in $\mathcal{P}$ have agreed upon a $CORE$, protocol St-Gen generates τ-$(1d)$-sharing in the following way: From the properties of $CORE$, the row polynomials of honest parties in $CORE$ define a unique degree-(τ, t) bivariate polynomial say $\overline{F}(x, y)$ with probability at least $(1 - \epsilon)$, such that each honest party P_i in $CORE$ possesses $\overline{f_i}(x) = \overline{F}(x, i)$. So the j^{th} point on $\overline{f_i}(x)$ polynomials, corresponding to all honest P_i's in $CORE$ define degree-t polynomial $\overline{p_j}(y) = \overline{F}(j, y)$. Furthermore, $|CORE| \geq 3t + 1$. So the parties in $CORE$ can enable each $P_j \in \mathcal{P}$ to privately reconstruct $\overline{p_j}(y)$ by applying standard OEC technique [35, 13]. Informally, OEC allows any party $P_\alpha \in \mathcal{P}$ to privately reconstruct a value v in asynchronous settings, which is shared among a set of parties $\overline{\mathcal{P}} \subseteq \mathcal{P}$ using a τ-degree polynomial, where $\tau < |\overline{\mathcal{P}}| - 2t$ [35, 13] (recall from Section 11.1.6). Once this is done, P_j can output $\overline{p_j}(0) = \overline{f_0}(j)$ as the j^{th} share of s, which is now τ-$(1d)$-shared using degree-τ polynomial $\overline{f_0}(x) = \overline{F}(x, 0)$.

Figure 11.4: Third Phase of protocol St-AVSS-Share: **Generation of τ-$(1d)$-sharing**

Protocol St-Gen($D, \mathcal{P}, s, \tau, \epsilon$)

Code for P_i: Every party executes this code

1. If $P_i \in CORE$, then for $j = 1, \ldots, n$, privately send $\overline{f_i}(j)$ to party P_j.
2. Apply OEC on $\overline{f_j}(i)$'s received from P_j's belonging to $CORE$ to privately reconstruct degree-t polynomial $\overline{p_i}(y)$ and hence $\overline{p_i}(0)$.
3. Output $s_i = \overline{f_0}(i) = \overline{p_i}(0)$ as the i^{th} share of D's secret and terminate St-Gen. D's secret $\overline{s} = \overline{f_0}(0)$ is now τ-$(1d)$-shared among the parties in $\mathcal{P}$ using degree-τ polynomial $\overline{f_0}(x)$.

Lemma 11.15 *Assuming that every honest party has agreed on $CORE$, St-Gen will generate τ-$(1d)$-sharing of of secret $\overline{s} = \overline{F}(0, 0)$ with probability at least $(1-\epsilon)$. If D is honest, then $\overline{s} = s$ where s is D's secret in St-Distr.*

PROOF: From Lemma 11.14, the row polynomials held by the honest parties in $CORE$ define a unique degree-(τ, t) bivariate polynomial, say $\overline{F}(x, y)$ with probability at least $(1 - \epsilon)$. This also implies that the i^{th} point on $\overline{f_j}(x)$ polynomials, corresponding to all honest P_j's in $CORE$ define degree-t polynomial $\overline{p_i}(y)$ for all $i = 1, \ldots, n$ with probability at least $(1 - \epsilon)$. So party P_i can apply OEC on $\overline{f_j}(i)$ values received from the parties in $CORE$ and correctly interpolate $\overline{p_i}(y)$ and obtain $\overline{p_i}(0)$ with probability at least $(1 - \epsilon)$. Now $\overline{f_0}(i) = \overline{p_i}(0)$ holds by the property of bivariate polynomial. So $\overline{s} = \overline{f_0}(0) = \overline{F}(0,0)$ is now τ-$(1d)$-shared using degree-τ polynomial $\overline{f_0}(x)$ where every honest $P_i \in \mathcal{P}$ holds $\overline{f_0}(i)$, the i^{th} share of s with probability at least $(1-\epsilon)$.

When D is honest then $\overline{F}(x,y) = F(x,y)$ and thus $\overline{s} = s$. Hence s will be τ-$(1d)$-shared among the parties using polynomial $f_0(x) = F(x,0)$. Notice that in this case the τ-$(1d)$-sharing will be generated without any error. □

11.2.4 Protocol St-AVSS: Statistical AVSS Sharing a Single Secret

Now our final statistical AVSS scheme is given in Fig. 11.5.

Figure 11.5: Protocol St-AVSS

Protocol St-AVSS-Share($D, \mathcal{P}, s, \tau, \epsilon$)

1. D executes St-Distr$(D, \mathcal{P}, s, \tau, \epsilon)$.
2. Every party P_i participates in St-Ver-Agree$(D, \mathcal{P}, s, \tau, \epsilon)$.
3. If $CORE$ is generated and agreed upon, then every P_i participates in St-Gen$(D, \mathcal{P}, s, \tau, \epsilon)$ and terminate St-AVSS-Share after terminating St-Gen.

Protocol St-AVSS-Rec($D, \mathcal{P}, s, \tau, \epsilon$)

i. Code for P_i:

1. Privately send s_i, the i^{th} share of the secret to every $P_j \in \mathcal{P}$.
2. Apply OEC on received s_j's, reconstruct the secret and terminate St-AVSS-Rec.

We now prove the properties of our statistical AVSS scheme.

Lemma 11.16 (AVSS-Termination) *Protocol St-AVSS satisfies* **termination** *property of Definition 11.2 with probability at least* $(1-\epsilon)$.

PROOF:**Termination 1:** We first prove **Termination 1** which says that if D is honest then every honest party must terminate St-AVSS-Share eventually. **Termination 1** will hold without any error probability. If D is honest then $AgreeSet^{P_\alpha}$ of every honest verifier P_α can eventually contain all the honest parties (at least $3t+1$) from $\mathcal{P}$. Thus $CORE$ can eventually contain all honest parties in $\mathcal{P}$ and will be accepted by every honest party. The rest now follows from the protocol steps, properties of $CORE$ and OEC.

Termination 2: We now prove **Termination 2** which says that if D is corrupted and some honest P_i has terminated St-AVSS-Share, then all honest parties will eventually terminate St-AVSS-Share. **Termination 2** will hold good with probability at least $(1-\epsilon)$. If D is corrupted and some honest P_i has terminated St-AVSS-Share, then it implies that P_i has received $CORE$ from the A-cast of D, checked its validity and accepted it. Moreover P_i has finally computed s_i

in protocol St-Gen. From the protocol steps of St-Ver-Agree and properties of A-cast every other honest party will also eventually accept $CORE$ after receiving it from the A-cast of D. Now from Lemma 11.14, the row polynomials held by the honest parties in $CORE$ define a unique degree-(τ, t) bivariate polynomial with probability at least $(1 - \epsilon)$. Hence in St-Gen, every other honest party P_j will compute s_j (the j^{th} share of secret) with probability at least $(1 - \epsilon)$ and terminate St-AVSS-Share with the same probability.

Termination 3: Finally, we proceed to prove **Termination 3** which says that if every honest party terminates St-AVSS-Share and starts St-AVSS-Rec, then every honest party will eventually terminate St-AVSS-Rec. **Termination 3** will hold with probability at least $(1-\epsilon)$. From Lemma 11.15, protocol St-Gen will generate correct τ-$(1d)$-sharing of a secret with probability at least $(1 - \epsilon)$. Now since the generated τ-$(1d)$-sharing is correct with probability at least $(1 - \epsilon)$, every party P_i will correctly reconstruct the secret in St-AVSS-Rec with probability at least $(1 - \epsilon)$. □

Lemma 11.17 (AVSS-Secrecy) *Protocol St-AVSS satisfies* **secrecy** *property of Definition 11.2.*

PROOF: We have to consider the case when D is honest. By Lemma 11.10, the polynomials $E^{(P_\alpha, \beta)}$ A-casted in St-Ver-Agree are completely random to $\mathcal{A}_t$ and hence can be ignored. Moreover at the end of St-AVSS-Share, a party P_i holds $f_i(x)$ and $p_i(y)$. Now by property of bivariate polynomial of degree-(τ, t), $\tau - t + 1$ coefficients of $F(x, y)$ will remain secure, where $F(x, y)$ is the polynomial used by D to hide his secret s. So $s = F(0, 0)$ will be secure from $\mathcal{A}_t$. □

Lemma 11.18 (AVSS-Correctness) *Protocol St-AVSS satisfies* **correctness** *property of Definition 11.2 with probability at least* $(1 - \epsilon)$.

PROOF: **Correctness 1:** We first prove **Correctness 1**. By **Termination 1** of St-AVSS (see Lemma 11.16), for an honest D, $CORE$ will always be agreed upon among all the honest parties in $\mathcal{P}$. Moreover, by Lemma 11.14 the row polynomials of the honest parties in $CORE$ define degree-(τ, t) bivariate polynomial $F(x, y)$ without any error when D is honest. Now by Lemma 11.15, secret $s = F(0, 0)$ will be τ-$(1d)$-shared among the parties in $\mathcal{P}$ without any error probability. Hence in St-AVSS-Rec, the parties will correctly reconstruct back s using OEC technique.

Correctness 2: If D is corrupted and the honest parties in $\mathcal{P}$ terminates St-AVSS-Share, then St-Gen has generated τ-$(1d)$-sharing of a secret $\overline{s}$ with probability at least $(1 - \epsilon)$. Since the generated τ-$(1d)$-sharing is correct with probability $(1 - \epsilon)$, in St-AVSS-Rec every party will reconstruct the secret with probability $(1 - \epsilon)$. □

Theorem 11.19 *Protocol St-AVSS consisting of sub-protocols (St-AVSS-Share, St-AVSS-Rec) constitutes a valid statistical AVSS scheme according to Definition 11.2.*

PROOF: Follows from Lemma 11.16, Lemma 11.17 and Lemma 11.18. □

Theorem 11.20 (Communication Complexity of St-AVSS)

- *Protocol St-AVSS-Share privately communicates $\mathcal{O}(n^3 \log(|\mathbb{F}|))$ and A-casts $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits, where $\log \frac{1}{\epsilon} = \log(|\mathbb{E}|)$.*
- *Protocol St-AVSS-Rec privately communicates $\mathcal{O}(n^2 \log(|\mathbb{F}|))$ bits.*

PROOF: In St-Distr, D privately communicates $\mathcal{O}((n\tau + n^3)\log(|\mathbb{F}|))$ bits. Since $t \leq \tau \leq 2t$, $\tau = \mathcal{O}(n)$. In St-Ver-Agree, the parties A-cast $\mathcal{O}(n^3 \log(|\mathbb{F}|) + n^2 \log \frac{1}{\epsilon})$ bits. In St-Gen and St-AVSS-Rec, parties privately communicate $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits. As $\log(|\mathbb{F}|) \leq \log \frac{1}{\epsilon}$, overall parties perform private communication of $\mathcal{O}(n^3 \log(|\mathbb{F}|))$ bits and A-cast of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits. □

11.3 Statistical AVSS For Sharing Multiple Secrets

We now present a statistical AVSS scheme, called St-AVSS-MS (here MS stands for multiple secrets) consisting of pair of protocols (St-AVSS-MS-Share,St-AVSS-MS-Rec). Protocol St-AVSS-MS-Share allows a dealer $D \in \mathcal{P}$ to generate τ-$(1d)$-sharing of secret $S = (s^1, \ldots, s^\ell)$, consisting of $\ell > 1$ elements from $\mathbb{F}$, where $t \leq \tau \leq 2t$. Notice that we assume ℓ to satisfy $\ell = poly(n, \log \frac{1}{\epsilon})$. Protocol St-AVSS-MS-Rec reconstructs the secret S, given its τ-$(1d)$-sharing.

Notice that we can generate τ-$(1d)$-sharing of S by concurrently executing protocol St-AVSS-Share (given in the previous section) ℓ times, once for each $s^l \in S$. This would require private communication of $\mathcal{O}(\ell n^3 \log(|\mathbb{F}|))$ bits and A-cast of $\mathcal{O}(\ell n^3 \log \frac{1}{\epsilon})$ bits. However, our protocol St-AVSS-MS-Share achieves the same task with a private communication of $\mathcal{O}((\ell n^2 + n^3)\log |\mathbb{F}|)$ bits and A-cast of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits. *It is to be noted that the A-cast communication of St-AVSS-MS-Share protocol is independent of ℓ.* This shows that executing a *single instance* of St-AVSS-MS-Share dealing with *multiple secrets* concurrently is advantageous over executing *multiple instances* of St-AVSS-Share dealing with *single secret*.

The structure of St-AVSS-MS-Share is divided into same three phases as in St-AVSS-Share. The corresponding protocols are St-Distr-MS, Single-Verifier-MS, St-Ver-Agree-MS and St-Gen-MS. Protocols St-Distr-MS and St-Gen-MS are straight forward extension of St-Distr and St-Gen respectively for ℓ values. Protocol Single-Verifier-MS (and hence St-Ver-Agree-MS) is also extension of Single-Verifier (and hence St-Ver-Agree) for ℓ values with the following difference: Instead of A-casting ℓ linear combination of polynomials corresponding to ℓ secrets, the dealer D A-casts *only one* linear combination of polynomials corresponding to all the ℓ secrets. In response, every party A-casts *only one* linear combination of values, instead of ℓ linear combination of values. *It is this crucial step, which results in the A-cast communication of St-AVSS-MS-Share to be independent of ℓ.* The protocols are given in Fig. 11.6, Fig. 11.7, Fig. 11.8, Fig. 11.9 and Fig. 11.10. The proofs for the properties of the protocols follow from the proofs of the protocols dealing with single secret (presented in previous section). For the sake of avoiding repetition, we do not provide any proof for these protocols. Rather, we give the final theorem stating that St-AVSS-MS is a valid statistical AVSS protocol and then present the theorem on the communication complexity of protocol St-AVSS-MS.

Theorem 11.21 *Protocol St-AVSS-MS consisting of sub-protocols (St-AVSS-MS-Share, St-AVSS-MS-Rec) constitutes a valid statistical AVSS scheme for ℓ secrets.*

Figure 11.6: First Phase of Protocol St-AVSS-MS-Share: **Distribution** Phase

Protocol St-Distr-MS($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \tau, \epsilon$**)**

Code for D**:** Only D executes this code

1. Select ℓ random, bivariate polynomials $F^1(x, y), \ldots, F^\ell(x, y)$ of degree-(τ, t) over $\mathbb{F}$, such that $F^l(0, 0) = s^l$ for $l = 1, \ldots, \ell$. Let $f_i^l(x) = F^l(x, i)$, $p_i^l(y) = F^l(i, y)$ for $0 \leq i \leq n$ and $l = 1, \ldots, \ell$.
2. Select $(t + 1)n$ degree-t, random, distinct *blinding polynomials* over $\mathbb{F}$, denoted by $b^{(P_j,1)}(y), \ldots, b^{(P_j,t+1)}(y)$ for $j = 1, \ldots, n$.
3. For $i = 1, \ldots, n$, send the following to party P_i:
   (a) Row polynomials $f_i^l(x)$ for $l = 1, \ldots, \ell$;
   (b) For $j = 1, \ldots, n$, the shares $b^{(P_j,1)}(i), \ldots, b^{(P_j,t+1)}(i)$.

Theorem 11.22 (Communication Complexity of St-AVSS-MS)

- *Protocol St-AVSS-MS-Share privately communicates* $\mathcal{O}((\ell n^2 + n^3) \log |\mathbb{F}|)$ *bits and A-casts* $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ *bits.*
- *Protocol St-AVSS-MS-Rec privately communicates* $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ *bits.*

PROOF: In St-Distr-MS, D privately communicates $\mathcal{O}((\ell n\tau + n^3) \log |\mathbb{F}|)$ bits. Since $t \leq \tau \leq 2t$, $\tau = \mathcal{O}(n)$. In St-Ver-Agree-MS, the parties A-cast $\mathcal{O}(n^3 \log |\mathbb{F}| + n^2 \log \frac{1}{\epsilon})$ bits. In St-Gen-MS and St-AVSS-MS-Rec, the parties privately communicate $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits. As $\log(|\mathbb{F}|) \leq \log \frac{1}{\epsilon}$, overall the protocol involves a private communication of $\mathcal{O}((\ell n^2 + n^3) \log |\mathbb{F}|)$ bits and A-cast of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits. □

11.4 A Different Interpretation of Protocol St-AVSS-MS

In St-AVSS-MS-Share, every secret s^l for $l = 1, \ldots, \ell$ is τ-$(1d)$-shared using degree-τ polynomial $f_0^l(x) = F^l(x, 0)$. Now by the **Secrecy** proof of St-AVSS, given in Lemma 11.17, we can claim that $(\tau + 1) - t$ coefficients of $f_0^l(x)$ are information theoretically secure for every $l = 1, \ldots, \ell$. This implies that St-AVSS-MS-Share shares $\ell(\tau + 1 - t)$ secrets with a private communication of $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits and A-cast of $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits. As the A-cast communication is independent of ℓ, we may ignore it and conclude that the amortized cost of sharing a single secret using St-AVSS-MS-Share is only $\mathcal{O}(n \log |\mathbb{F}|)$. This is because by setting $\tau = 2t$ (which is the maximum value of τ), we see that St-AVSS-MS-Share can share $\ell(t + 1) = \Theta(\ell n)$ secrets by privately communicating $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits. Now putting it in other way, D can share $\ell(t + 1)$ secrets using St-AVSS-MS-Share by choosing a random polynomial $f_0^l(x)$ (of degree $\tau = 2t$) with lower order $t + 1$ coefficients as secrets and then choosing a random degree-(τ, t) bivariate polynomial $F^l(x, y)$ with $F^l(x, 0) = f_0^l(x)$ for $l = 1, \ldots, \ell$ and finally executing St-AVSS-MS-Share with $F^1(x, y), \ldots, F^\ell(x, y)$.

Figure 11.7: Steps to be executed with respect to a Single Verifier for multiple secrets

Protocol Single-Verifier-MS($V, \mathcal{P}, S, \tau, \epsilon$)

i. Code for P_i: Every party in $\mathcal{P}$, including D and V, executes this code.

1. Wait to receive the following from D:
   (a) For $l = 1, \ldots, \ell$, the row polynomial $\overline{f_i^l}(x)$;
   (b) For $j = 1, \ldots, n$, the shares $b^{(P_j,1)}(i), \ldots, b^{(P_j,t+1)}(i)$.
2. After receiving, check whether $\overline{f_i^l}(x)$ is a degree-τ polynomial for all $l = 1, \ldots, \ell$. If yes, then privately send a `Received-From-D` signal to V.

ii. Code for V: Only V executes this code.

1. Wait to obtain `Received-From-D` signal from $3t+1$ parties. Put the identities of these $3t+1$ parties in a set $ReceivedSet^{(V,1)}$. Select a random value $r^{(V,1)} \in \mathbb{E}$ and A-cast $(r^{(V,1)}, ReceivedSet^{(V,1)})$.
2. After the previous step, for every new receipt of `Received-From-D` signal from $P_\alpha \notin ReceivedSet^{(V,\beta-1)}$ such that $1 < \beta \leq t+1$, construct $ReceivedSet^{(V,\beta)} = ReceivedSet^{(V,\beta-1)} \cup \{P_\alpha\}$, select a random $r^{(V,\beta)} \in \mathbb{E} \setminus \{r^{(V,1)}, \ldots, r^{(V,\beta-1)}\}$ and A-cast $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$.

iii. Code for D: Only D executes this code.

1. If $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$ is received from the A-cast of V, then A-cast the polynomial $E^{(V,\beta)}(y) = \mathsf{LCP}(\mathcal{E}, R)$. Here $\mathcal{E} = \{b^{(V,\beta)}(y), p_1^1(y), \ldots, p_n^1(y), \ldots, p_1^\ell(y), \ldots, p_n^\ell(y)\}$ and $R = (1, r^{(V,\beta)}, \left(r^{(V,\beta)}\right)^2, \ldots, \left(r^{(V,\beta)}\right)^{\ell n})$.

iv. Code for P_i: Every party in $\mathcal{P}$ executes this code.

1. If $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$ is received from the A-cast of V, then do the following:
   (a) Check if $P_i \in ReceivedSet^{(V,\beta)}$. If yes, then A-cast $e_i^{(V,\beta)} = \mathsf{LCV}(\Delta_i, R)$, where $\Delta_i = \{b^{(V,\beta)}(i), \overline{f_i^1}(1), \ldots, \overline{f_i^1}(n), \ldots, \overline{f_i^\ell}(1), \ldots, \overline{f_i^\ell}(n)\}$ and $R = (1, r^{(V,\beta)}, \left(r^{(V,\beta)}\right)^2, \ldots, \left(r^{(V,\beta)}\right)^{\ell n})$.
2. Say that **party P_j agrees with D with respect to** $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$ if all the following holds:
   (a) $E^{(V,\beta)}(y)$ is a degree-t polynomial,
   (b) $P_j \in ReceivedSet^{(V,\beta)}$ and
   (c) $e_j^{(V,\beta)} = E^{(V,\beta)}(j)$ where $e_j^{(V,\beta)}$, $E^{(V,\beta)}(y)$ and $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$ are received from the A-casts of P_j, D and V respectively.
3. With respect to $(r^{(V,\beta)}, ReceivedSet^{(V,\beta)})$, when there are $3t+1$ P_j's who agree with D, add all of them in a set $AgreeSet^{(V,\beta)}$.

Figure 11.8: Second Phase of Protocol St-AVSS-MS-Share: **Verification & Agreement on CORE** phase

Protocol St-Ver-Agree-MS($D, \mathcal{P}, S, \tau, \epsilon$)

Here, in step i(1) of St-Ver-Agree, P_i invokes Single-Verifier-MS($P_\alpha, \mathcal{P}, S, \tau, \epsilon$) instead of Single-Verifier. The rest of the steps are same as in Protocol St-Ver-Agree.

Figure 11.9: Third Phase of protocol St-AVSS-MS-Share: **Generation of τ-($1d$)-sharing** Phase

Protocol St-Gen-MS($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \tau, \epsilon$)

Code for P_i:

1. If $P_i \in CORE$, then privately send $\overline{f_i^l}(j)$ to party P_j for $l = 1, \ldots, \ell$ and $j = 1, \ldots, n$.
2. Apply OEC on $\overline{f_j^l}(i)$'s received from P_j's belonging to $CORE$ to privately reconstruct degree-t polynomials $\overline{p_i^l}(y)$'s, for $l = 1, \ldots, \ell$.
3. For $l = 1, \ldots, \ell$, output $s_i^l = \overline{f_0^l}(i) = \overline{p_i^l}(0)$ as the i^{th} share of D's secret $\overline{s^l}$ and terminate St-Gen-MS. D's secret $\overline{s^l} = \overline{f_0^l}(0)$ is now τ-($1d$)-shared using degree-τ polynomial $\overline{f_0^l}(x)$.

11.5 Finding (n,t)-star Structure in a Graph

We now describe an existing solution for a graph theoretic problem, called finding (n,t)-star in an undirected graph $G = (V, E)$. Our perfect AVSS protocol exploits several interesting properties of (n,t)-star. An (n,t)-star in $G = (V, E)$ with $V = \mathcal{P} = \{P_1, \ldots, P_n\}$ is defined as follows:

Definition 11.23 (**(n,t)-star[35, 19]**) *Let G be an undirected graph with the n parties in $\mathcal{P}$ as its vertex set. We say that a pair $(\mathcal{C}, \mathcal{D})$ of sets with $\mathcal{C} \subseteq \mathcal{D} \subseteq \mathcal{P}$ is an (n,t)-star in G, if the following hold:*

1. *$|\mathcal{C}| \geq n - 2t$;*
2. *$|\mathcal{D}| \geq n - t$;*
3. *For every $P_j \in \mathcal{C}$ and every $P_k \in \mathcal{D}$ the edge (P_j, P_k) exists in G.*

Ben-Or et al. [19] have presented an elegant and efficient algorithm for finding an (n,t)-star in a graph of n nodes, *provided that the graph contains a clique of size $n - t$*. The algorithm, called Find-STAR outputs either an (n,t)-star or the message `star-Not-Found`. Whenever the input graph contains a clique of size $n - t$, Find-STAR always outputs an (n,t)-star in the graph.

Figure 11.10: Protocol St-AVSS-MS

Protocol St-AVSS-MS-Share($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \tau, \epsilon$)

1. D executes S-Distr-MS($D, \mathcal{P}, s, \tau, \epsilon$).
2. Every party P_i participates in St-Ver-Agree-MS($D, \mathcal{P}, s, \tau, \epsilon$).
3. If $CORE$ is agreed upon then every P_i participates in St-Gen-MS($D, \mathcal{P}, s, \tau, \epsilon$) and terminate St-AVSS-MS-Share after terminating St-Gen-MS.

Protocol St-AVSS-MS-Rec($D, \mathcal{P}, S, \tau, \epsilon$)

i. Code for P_i:

1. For $l = 1, \ldots, \ell$, privately send s_i^l, the i^{th} share of $\overline{s^l}$ to every $P_j \in \mathcal{P}$.
2. For $l = 1, \ldots, \ell$, apply OEC on received s_j^l's, reconstruct $\overline{s^l}$ and terminate St-AVSS-MS-Rec.

Actually, algorithm Find-STAR takes the complementary graph $\overline{G}$ of G as input and tries to find (n,t)-$\overline{\text{star}}$ in $\overline{G}$ where (n,t)-$\overline{\text{star}}$ is a pair $(\mathcal{C}, \mathcal{D})$ of sets with $\mathcal{C} \subseteq \mathcal{D} \subseteq \mathcal{P}$, satisfying the following conditions:

1. $|\mathcal{C}| \geq n - 2t$;
2. $|\mathcal{D}| \geq n - t$;
3. There are no edges between the nodes in $\mathcal{C}$ and nodes in $\mathcal{C} \cup \mathcal{D}$ in $\overline{G}$.

Clearly, a pair $(\mathcal{C}, \mathcal{D})$ representing an (n,t)-$\overline{\text{star}}$ in $\overline{G}$, is an (n,t)-star in G. Recasting the task of Find-STAR in terms of complementary graph $\overline{G}$, we say that Find-STAR outputs either an (n,t)-$\overline{\text{star}}$, or a message `star-Not-Found`. Whenever, *the input graph $\overline{G}$ contains an independent set of size $n - t$*, Find-STAR always outputs an (n,t)-$\overline{\text{star}}$. For simple notation, we denote $\overline{G}$ by H. The algorithm Find-STAR is presented in Fig. 11.11 and its properties are presented in the sequel for ready access. The proofs recorded below are taken from [35].

Lemma 11.24 ([35]) *If Find-STAR outputs $(\mathcal{C}, \mathcal{D})$ on input graph H, then $(\mathcal{C}, \mathcal{D})$ is a (n,t)-$\overline{\text{star}}$ in H.*

PROOF: Clearly, if Find-STAR outputs $(\mathcal{C}, \mathcal{D})$ then $|\mathcal{C}| \geq n - 2t$ and $|\mathcal{D}| \geq n - t$, and $\mathcal{C} \subseteq \mathcal{D}$. We now show that for every $P_i \in \mathcal{C}$ and every $P_j \in \mathcal{D}$, the nodes P_i and P_j are not neighbors in H.

On the contrary, assume that $P_i \in \mathcal{C}$ and $P_j \in \mathcal{D}$, such that (P_i, P_j) is an edge in H. As $P_j \in \mathcal{D}$, we must have $P_j \notin B$. By the definition of B, we have $P_j \notin N$ (if $P_i \in \mathcal{C}$ and $P_j \in N$, then $P_j \in B$). Furthermore, $P_i \in \mathcal{C} \subseteq \overline{N}$. Thus, both P_i and P_j are unmatched. Consequently, the edge (P_i, P_j) can be added to the maximum matching to create a larger matching, which is a contradiction. □

Figure 11.11: Algorithm For Finding (n, t)-star

.

Algorithm Find-STAR(H)

1. Find a maximum matching M in H. Let N be the set of matched nodes (namely, the endpoints of the edges in M), and let $\overline{N} = \mathcal{P} \setminus N$.

2. Compute output as follows (which could be either (n, t)-$\overline{\text{star}}$ or a message `star-Not-Found`):

   (a) Let $T = \{P_i \in \overline{N} | \exists P_j, P_k \text{ s.t } (P_j, P_k) \in M \text{ and } (P_i, P_j), (P_i, P_k) \in E\}$. T is called the set of triangle-heads.

   (b) Let $\mathcal{C} = \overline{N} \setminus T$.

   (c) Let B be the set of matched nodes that have neighbors in $\mathcal{C}$. So $B = \{P_j \in N | \exists P_i \in \mathcal{C} \text{ s. t. } (P_i, P_j) \in E\}$.

   (d) Let $\mathcal{D} = \mathcal{P} \setminus B$. If $|\mathcal{C}| \geq n-2t$ and $|\mathcal{D}| \geq n-t$, output $(\mathcal{C}, \mathcal{D})$. Otherwise, output `star-Not-Found`.

Lemma 11.25 ([35]) *Let H be a graph with $\mathcal{P}$ as its vertex set, containing an independent set of size $n - t$. Then algorithm Find-STAR always outputs a (n, t)-$\overline{\text{star}}$, say $(\mathcal{C}, \mathcal{D})$, in H.*

Proof: We show that if H contains an independent set of size $n - t$, then Find-STAR can always find $|\mathcal{C}|$ and $|\mathcal{D}|$ to be large enough (i.e $|\mathcal{C} \geq n - 2t|$ and $|\mathcal{D}| \geq n - t$) to output a (n, t)-$\overline{\text{star}}$ $(\mathcal{C}, \mathcal{D})$.

We first show that $|\mathcal{C}| \geq n - 2t$. Let $I \subseteq \mathcal{P}$ be an independent set in H, and let $\overline{I} = \mathcal{P} \setminus I$. Since the size of I is $n - t$, we have $|\overline{I}| \leq t$. Let $F = I \setminus C$. We show that $|F| \leq |\overline{I}|$. Consequently, we have $|\mathcal{C}| \geq |I| - |F| \geq n - 2t$. To prove that $|F| \leq |\overline{I}|$, we show a one-to-one correspondence $\phi : F \rightarrow \overline{I}$. Let $P_i \in F$. Since $P_i \notin \mathcal{C}$, we have either $P_i \in N$ or $P_i \in T$.

Case I: $P_i \in N$. Then let $\phi(P_i)$ be the node matched to P_i in M. Clearly, $\phi(P_i) \in \overline{I}$: otherwise, we had an edge $(P_i, \phi(P_i))$ where both P_i and $\phi(P_i)$ are in an independent set.

Case II: $P_i \in T$. By the definition of T, node P_i has two neighbors, P_j and P_k, such that $(P_j, P_k) \in M$. Arbitrarily set $\phi(P_i) = P_j$. Clearly, both P_j and P_k are in $\overline{I}$.

We now show that ϕ is one-to-one. Consider two distinct nodes, P_l and P_m from F. We have three cases:

Case 1: $P_l, P_m \in N$. In this case, $\phi(P_l) \neq \phi(P_m)$ since M is a matching.

Case 2: $P_l \in N$ and $P_m \in T$. Since $P_m \in T$, there exists an edge between P_m and the node matched to $\phi(P_m)$. Since, $P_l \in N$, the node matched to $\phi(P_l)$ is P_l. Now assume that $\phi(P_l) = \phi(P_m)$. Thus, (P_l, P_m) is an edge in H, which is a contradiction, as P_l and P_m are in the independent set I.

Case 3: $P_l, P_m \in T$. Assume $\phi(P_l) = \phi(P_m)$. Let P_a be the node matched to $\phi(P_m)$ in M. Both P_l and P_m are neighbors of both $\phi(P_m)$ and P_a. However, in this case the matching M is not maximum since, for instance, $M \setminus \{(\phi(P_m), P_a)\} \cup \{(\phi(P_m), P_l), (P_a, P_m)\}$ is a larger matching.

Now, it remains to show that $|\mathcal{D}| \geq n - t$. Recall that $\mathcal{D} = \mathcal{P} \setminus B$. We show that $|B| \leq |M|$. Since H contains an independent set of size $n - t$, we have $|M| \leq t$. Thus, $|\mathcal{D}| = n - |B| \geq n - |M| \geq n - t$. To prove $|B| \leq |M|$, we show that at most one of the endpoints of every edge $(P_a, P_b) \in M$ is in B. On the contrary let both P_a and P_b have neighbors in C, and let $P_c, P_d \in C$ be the neighbors of P_a and P_b, respectively. Surely, $P_c \neq P_d$ (otherwise, P_c was a triangle-head and we had $P_c \in \mathcal{C}$). However, in this case M is not maximum, since, $M \setminus \{(P_a, P_b)\} \cup \{(P_a, P_c), (P_b, P_d)\}$ is a certainly larger matching. □

11.6 Perfect AVSS for Sharing a Single Secret

We now present a novel AVSS scheme, called Pf-AVSS (perfect AVSS), consisting of pair of protocols, (Pf-AVSS-Share,Pf-AVSS-Rec). The protocol Pf-AVSS-Share allows a dealer $D \in \mathcal{P}$ (dealer can be any party from $\mathcal{P}$) to τ-$(1d)$-share a single secret from $\mathbb{F}$, among the parties in $\mathcal{P}$, where $t \leq \tau \leq 2t$. Protocol Pf-AVSS-Rec allows the parties in $\mathcal{P}$ to reconstruct the secret, given its τ-$(1d)$-sharing. The structure of Pf-AVSS-Share is divided into a sequence of following three phases. If D is honest then eventually all the three phases will be terminated by all honest parties in $\mathcal{P}$. The sharing phase of our statistical AVSS protocol St-AVSS-Share presented earlier in this chapter is also structured into the same three phases. However, the implementation of all the three phases of our perfect AVSS is completely different from that of statistical AVSS. More importantly, the last two phases of the statistical AVSS involves a negligible error probability, whereas the implementation of all the three phases are perfect (error free) in all respects for our perfect AVSS.

1. **Distribution Phase:** As the name suggests, in this phase, D on having a secret s, distributes information to the parties in $\mathcal{P}$.
2. **Verification & Agreement on CORE Phase:** Here parties jointly perform some computation and communication in order to verify consistency of the information distributed by D in **Distribution** Phase. In case of successful verification, all honest parties agree on a set of at least $3t+1$ parties called $CORE$, satisfying certain property (mentioned in the sequel).
3. **Generation of τ-$(1d)$-sharing Phase:** If $CORE$ is agreed upon in the previous phase, then here every party performs local computation on the data received (during **Verification & Agreement on CORE Phase**) from the parties in $CORE$ to finally generate the τ-$(1d)$-sharing of secret s.

An honest party will terminate Pf-AVSS-Share, if it successfully completes the last phase, namely **Generation of τ-$(1d)$-sharing** Phase. If D is honest then each honest party will eventually terminate the last phase. Moreover, if D is corrupted and some honest party terminates the last phase, then each honest party will also eventually terminate the last phase (and hence Pf-AVSS-Share).

We now focus on the details of each of the aforementioned phases of protocol Pf-AVSS-Share in order.

11.6.1 Distribution Phase

Here D on having a secret s, selects a random bivariate polynomial $F(x, y)$ of degree-(τ, t) (i.e., the degree of the polynomial in x is τ and the degree of the polynomial in y is t), such that $F(0, 0) = s$ and sends $f_i(x) = F(x, i)$ and $p_i(y) = F(i, y)$ to party P_i. We will call the degree-τ $f_i(x)$ polynomials as *row polynomials* and degree-t $p_i(y)$ polynomials as *column polynomials*. The protocol is given in Fig. 11.12.

Figure 11.12: First Phase of Protocol Pf-AVSS-Share: **Distribution by** D Phase

> **Protocol Pf-Distr($D, \mathcal{P}, s, \tau$)**
>
> **Code for D:** Only D executes this code
>
> 1. Select a random bivariate polynomial $F(x, y)$ of degree-(τ, t) over $\mathbb{F}$, such that $F(0, 0) = s$.
> 2. Send $f_i(x) = F(x, i)$ and $p_i(y) = F(i, y)$ to party P_i, for $i = 1, \ldots, n$.

Notice that unlike protocol St-Distr (used in St-AVSS presented in Section 11.2), D gives *both* row and column polynomials to respective parties in protocol Pf-Distr. Also D does not use *blinding polynomials*, as in St-Distr. The blinding polynomials were used in St-AVSS to probabilistically check the consistency of the information distributed by D. However, Pf-AVSS being a perfect AVSS scheme, we do not use any probabilistic checks to verify the consistency of the information distributed by D. Rather we employ completely different mechanism, as will be shown in the sequel.

In the next section, we describe **Verification & Agreement on CORE** Phase. If this phase is successful, then at the end of **Generation of τ-($1d$)-sharing** Phase, the secret s will be τ-($1d$)-sharing among the parties using degree-τ polynomial $f_0(x) = F(x, 0)$.

11.6.2 Verification & Agreement on CORE Phase

The goal of this phase is to check the existence of a set of parties called $CORE$. If a $CORE$ exists then every honest party will agree on $CORE$, where $CORE$ is defined as follows

Definition 11.26 (Property of $CORE$:) *$CORE$ is a set of at least $3t+1$ parties such that the row polynomials (received in* **Distribution** *Phase) of the honest parties in $CORE$ define a unique bivariate polynomial say, $\overline{F}(x, y)$ of degree-(τ, t). Moreover, if D is honest, then $\overline{F}(x, y) = F(x, y)$, where $F(x, y)$ was chosen by D in* **Distribution** *Phase.*

The property of $CORE$ ensures that for every $j \in \{1, \ldots, n\}$, the j^{th} point on row polynomials of honest parties in $CORE$ define degree-t column polynomial $\overline{p_j}(y) = \overline{F}(j, y)$. So once $CORE$ is constructed and agreed upon by each honest party then $\overline{p_j}(0)$ can be privately reconstructed by P_j with the help of the parties in $CORE$ by using OEC [35]. This will generate τ-($1d$)-sharing of $\overline{s} = \overline{F}(0, 0)$, where $\overline{s}$ will be τ-$(1d)$-sharing using degree-τ polynomial $\overline{f_0}(x) = \overline{F}(x, 0)$ and each (honest) P_j will have his share $\overline{f_0}(j) = \overline{p_j}(0)$ of $\overline{s}$. Moreover, if D is honest, then $\overline{s} = s$ as $\overline{F}(x, y) = F(x, y)$. Note that even though the degree of row polynomials is more than t (if $\tau > t$), we create a situation where parties need not have to reconstruct them. To obtain the shares corresponding to τ-$(1d)$-sharing of s, the parties need to reconstruct degree-t column polynomials only. *This is one of the crucial steps of our AVSS.*

Notice that in our statistical AVSS also, $CORE$ has same properties as above. However, in our statistical AVSS, we used probabilistic checks to check the consistency of the information delivered by D and generated $CORE$ with high probability. However, in our perfect AVSS, we check the consistency of the information delivered by D without using any probabilistic checks. In fact, we use completely different techniques. As a result, we can generate $CORE$ without any error. We now give an outline of this phase.

Outline of Current Phase: Here the parties upon receiving row and column polynomials (from D), interact with each other to check the consistency of their common values (on their polynomials). After successfully verifying the consistency, parties A-cast OK signals. Using these signals, a graph with the parties as vertex set is formed and applying Find-STAR on the graph, a sequence of distinct (n, t)-stars are obtained. The reason for constructing a sequence of (n, t)-stars will be clear in the sequel.

The row polynomials of the honest parties in $\mathcal{C}$ component of each (n, t)-star in the above graph defines a unique bivariate polynomial of degree-(τ, t). For every generated (n, t)-star, D tries to find whether $CORE$ can be generated from it. The generation process of $CORE$ attempts to use several interesting features of (n, t)-star (mainly its $\mathcal{C}$ component). Specifically, we show that:

(a) If D is honest and $\mathcal{C}$ component of some (n, t)-star $(\mathcal{C}, \mathcal{D})$ contains at least $2t+1$ honest parties, then $CORE$ will be eventually generated from $(\mathcal{C}, \mathcal{D})$;

(b) If D is honest, then eventually some (n, t)-star $(\mathcal{C}, \mathcal{D})$ will be generated, where $\mathcal{C}$ will contain at least $2t + 1$ honest parties. However, dealer D may not know which (n, t)-star it is.

The above two important properties of (n, t)-star in our context are at the heart of our perfect AVSS protocol. In addition to this, we also show the following:

(c) If $CORE$ is generated from some (n, t)-star $(\mathcal{C}, \mathcal{D})$ (irrespective of whether D is honest or corrupted), then the row polynomials of the honest parties in $CORE$, as well as the honest parties in $\mathcal{C}$ define the same bivariate polynomial of degree-(τ, t).

To check whether $CORE$ can be generated from an (n, t)-star $(\mathcal{C}, \mathcal{D})$, we use the following idea: First of all, we note that the following holds for the honest parties in $\mathcal{C}$ (even though the identities of the honest are not known): the row polynomials of the honest parties in $\mathcal{C}$ define a unique bivariate polynomial of degree-(τ, t), say $\overline{F}(x, y)$. Then we try to find out how many other parties are there whose row polynomials also lie on $\overline{F}(x, y)$. For this, we first list all such P_j's whose column polynomial is pairwise consistent with the row polynomial of at least $2t + 1$ parties in $\mathcal{C}$. This we can do by finding all such P_j's, who have at least $2t + 1$ neighbors in $\mathcal{C}$. All such P_j's are put in a list $\mathcal{F}$. Informally, the column polynomial of each P_j in $\mathcal{F}$ will lie on $\overline{F}(x, y)$. This is because, P_j's column polynomial is of degree-t and it is consistent with row polynomials of at least $t+1$ *honest* parties in $\mathcal{C}$, where the row polynomials define $\overline{F}(x, y)$.

We next list all such P_j's, whose row polynomial is pair-wise consistent with column polynomial of at least $\tau+t+1$ parties in $\mathcal{F}$. This can be done by finding all such P_j's, who have at least $\tau+t+1$ neighbors in $\mathcal{F}$. We put all such P_j's in a list $\mathcal{E}$. Informally, the row polynomial of each P_j in $\mathcal{E}$ will lie on $\overline{F}(x, y)$. This is because, P_j's row polynomial is of degree-τ and it is consistent with column polynomials of at least $\tau+1$ *honest* parties in $\mathcal{F}$, where the column polynomials define $\overline{F}(x, y)$. If both $\mathcal{E}$ and $\mathcal{F}$ contain at least (probably different) $3t+1$ parties, then we assign $\mathcal{E}$ as $CORE$.

The above process of generating $\mathcal{E}$ and $\mathcal{F}$ from the corresponding (n, t)-star is done for many distinct (n, t)-stars that are present in the graph. The generation of many (n, t)-stars in our case is essential as the $\mathcal{C}$ component of the first (n, t)-star may not contain at least $2t+1$ honest parties and hence may never lead to $CORE$ (by employing the above mechanism). *This implies that if our protocol stops after generating the first (n, t)-star then the protocol may not terminate even for an honest D.* However, as mentioned earlier, if D is honest and if we keep on generating a sequence of distinct (n, t)-stars after every update in the graph, then eventually a (n, t)-star will be generated, whose $\mathcal{C}$ component will contain at least $2t+1$ honest parties (and hence by employing the above mechanism we can generate $CORE$). Moreover, we will show that this will not take infinite iterations, as the maximum number of distinct (n, t)-stars that can generated in the graph is $\mathcal{O}(n^2)$.

Finally, before presenting the protocol for this phase, we stress that existing AVSS of [19] need not generate a sequence of (n, t)-stars because it has to generate only t-$(1d)$-sharing. Hence the AVSS of [19] stops after generating the first (n, t)-star and then using the $\mathcal{D}$ component of the generated (n, t)-star, it could generate t-$(1d)$-sharing of s (for details see [35]). The steps of this phase are given in protocol Pf-Ver-Agree that appear in Fig. 11.13.

We now prove the properties of protocol Pf-Ver-Agree.

Lemma 11.27 *For any (n, t)-star $(\mathcal{C}, \mathcal{D})$ in graph G_k of honest P_k, the row polynomials held by honest parties in $\mathcal{C}$ define a unique bivariate polynomial of degree-(τ, t), say $\overline{F}(x, y)$, such that column polynomial $\overline{p_j}(y)$ held by every honest $P_j \in \mathcal{D}$ satisfies $\overline{p_j}(y) = \overline{F}(j, y)$. Moreover, if D is honest, then $\overline{F}(x, y) = F(x, y)$.*

PROOF: For any (n, t)-star $(\mathcal{C}, \mathcal{D})$, $|\mathcal{C}| \geq n - 2t$ and $|\mathcal{D}| \geq n - t$. So $\mathcal{C}$ and $\mathcal{D}$ contain at least $n - 3t \geq t + 1$ and $n - 2t \geq 2t + 1$ honest parties, respectively. Let l and m be the number of honest parties in $\mathcal{C}$ and $\mathcal{D}$ respectively where $l \geq t + 1$ and $m \geq 2t + 1$. Without loss of generality, we assume $P_1, \ldots, P_l$, respectively $P_1, \ldots, P_m$ are the set of honest parties in $\mathcal{C}$ and $\mathcal{D}$. Now by the construction of (n, t)-star, for every pair of honest parties (P_i, P_j) with $P_i \in \mathcal{C}$ and $P_j \in \mathcal{D}$, the row polynomial $\overline{f_i}(x)$ of honest P_i and the column polynomial $\overline{p_j}(y)$ of honest P_j satisfy $\overline{f_i}(j) = \overline{p_j}(i)$. We now claim that the above statement implies that there exists a unique bivariate polynomial $\overline{F}(x, y)$ of degree-(τ, t), such that for $i = 1, \ldots, l$, we have $\overline{F}(x, i) = \overline{f_i}(x)$ and for $j = 1, \ldots, m$, we have $\overline{F}(j, y) = \overline{p_j}(y)$. The proof is similar to the proof of Lemma 11.9. □

Lemma 11.28 *For an honest D, an (n, t)-star $(\mathcal{C}^\beta, \mathcal{D}^\beta)$ with $\mathcal{C}^\beta$ containing at least $2t+1$ honest parties will be generated eventually.*

Figure 11.13: Second Phase of Protocol Pf-AVSS-Share: **Verification & Agreement on CORE** phase

Protocol Pf-Ver-Agree($D, \mathcal{P}, s, \tau$)

i. Code for P_i: Every party $P_i \in \mathcal{P}$ (including D) executes this code.

1. Wait to receive row polynomial $\overline{f_i}(x)$ of degree-τ and column polynomial $\overline{p_i}(y)$ of degree-t from D. Upon receiving, send $\overline{f_{ij}} = \overline{f_i}(j)$ and $\overline{p_{ij}} = \overline{p_i}(j)$ to party P_j, for $j = 1, \ldots, n$.
2. Upon receiving $\overline{f_{ji}}$ and $\overline{p_{ji}}$ from P_j, check if $\overline{f_i}(j) \stackrel{?}{=} \overline{p_{ji}}$ and $\overline{p_i}(j) \stackrel{?}{=} \overline{f_{ji}}$. If both the equalities hold, A-cast $\texttt{OK}(P_i, P_j)$.
3. Construct an undirected graph G_i with $\mathcal{P}$ as vertex set. Add an edge (P_j, P_k) in G_i upon receiving (a) $\texttt{OK}(P_k, P_j)$ from the A-cast of P_k and (b) $\texttt{OK}(P_j, P_k)$ from the A-cast of P_j.

ii. Code for D: Only D executes this code.

1. For every new receipt of some $\texttt{OK}(*, *)$ update G_D. If a new edge is added to G_D, then execute Find-STAR($\overline{G_D}$). Let there are $\alpha \geq 0$ distinct (n, t)-stars that are found in the past from different executions of Find-STAR($\overline{G_D}$).
   (a) Now if an (n, t)-star is found from the current execution of Find-STAR($\overline{G_D}$) that is distinct from all the previous α (n, t)-stars obtained before, do the following:
      i. Call the new (n, t)-star as $(\mathcal{C}^{\alpha+1}, \mathcal{D}^{\alpha+1})$.
      ii. Create a list $\mathcal{F}^{\alpha+1}$ as follows: Add P_j to $\mathcal{F}^{\alpha+1}$ if P_j has at least $2t + 1$ neighbors in $\mathcal{C}^{\alpha+1}$ in G_D.
      iii. Create a list $\mathcal{E}^{\alpha+1}$ as follows: Add P_j to $\mathcal{E}^{\alpha+1}$ if P_j has at least $\tau + t + 1$ neighbors in $\mathcal{F}^{\alpha+1}$ in G_D.
      iv. For every γ, with $\gamma = 1, \ldots, \alpha$ update $\mathcal{F}^{\gamma}$ and $\mathcal{E}^{\gamma}$:
         A. Add P_j to $\mathcal{F}^{\gamma}$, if $P_j \notin \mathcal{F}^{\gamma}$ and P_j has at least $2t + 1$ neighbors in $\mathcal{C}^{\gamma}$ in G_D.
         B. Add P_j to $\mathcal{E}^{\gamma}$, if $P_j \notin \mathcal{E}^{\gamma}$ and P_j has at least $\tau + t + 1$ neighbors in $\mathcal{F}^{\gamma}$ in G_D.
   (b) If no (n, t)-star is found or an (n, t)-star that has been already found in the past is obtained, then execute step (a).iv(A-B) to update existing $\mathcal{F}^{\gamma}$'s and $\mathcal{E}^{\gamma}$'s.
   (c) Now let β be the first index among already generated $\{(\mathcal{E}^1, \mathcal{F}^1), \ldots, (\mathcal{E}^{\delta}, \mathcal{F}^{\delta})\}$ such that both $\mathcal{E}^{\beta}$ and $\mathcal{F}^{\beta}$ contains at least $3t + 1$ parties (Note that if step (a) is executed, then $\delta = \alpha + 1$; else $\delta = \alpha$). Assign $CORE = \mathcal{E}^{\beta}$ and A-cast $((\mathcal{C}^{\beta}, \mathcal{D}^{\beta}), (\mathcal{E}^{\beta}, \mathcal{F}^{\beta}))$.

iii. Code for P_i: Every party $P_i \in \mathcal{P}$ (including D) executes this code.

1. Wait to receive $((\mathcal{C}^{\beta}, \mathcal{D}^{\beta}), (\mathcal{E}^{\beta}, \mathcal{F}^{\beta}))$ from the A-cast of D, such that both $\mathcal{E}^{\beta}$ and $\mathcal{F}^{\beta}$ contains at least $3t + 1$ parties.
2. Wait until $(\mathcal{C}^{\beta}, \mathcal{D}^{\beta})$ becomes a valid (n, t)-star in G_i.
3. Wait until every party $P_j \in \mathcal{F}^{\beta}$ has at least $2t + 1$ neighbors in $\mathcal{C}^{\alpha}$ in G_i.
4. Wait until every party $P_j \in \mathcal{E}^{\beta}$ has at least $\tau + t + 1$ neighbors in $\mathcal{F}^{\alpha}$ in G_i.
5. Accept $CORE = \mathcal{E}^{\beta}$.

PROOF: For an honest D, eventually the edges between each pair of honest parties will vanish from the complementary graph $\overline{G_D}$. So the edges in $\overline{G_D}$ will be either (a) between an honest and a corrupted party OR (b) between a corrupted and another corrupted party. Let β be the first index, such that (n, t)-star $(\mathcal{C}^{\beta}, \mathcal{D}^{\beta})$ is generated in $\overline{G_D}$, when $\overline{G_D}$ contains edges of above two types only. Now, by construction of $\mathcal{C}^\beta$ (see Algorithm Find-STAR), it excludes the parties in N (set of parties that are endpoints of the edges of maximum matching M) and T (set of parties that are triangle-head). An honest P_i belonging to N implies that $(P_i, P_j) \in M$ for some P_j and hence P_j is corrupted (as the current $\overline{G_D}$ does not have edge between two honest parties). Similarly, an honest party P_i belonging to T implies that there is some $(P_j, P_k) \in M$ such that (P_i, P_j) and (P_j, P_k) are edges in $\overline{G_D}$. This clearly implies that both P_j and P_k are surely corrupted. So for every honest P_i outside $\mathcal{C}^\beta$, at least one (if P_i belongs to N, then one; if P_i belongs to T, then two) corrupted party also remains outside $\mathcal{C}^\beta$. As there are at most t corrupted parties, C^β may exclude at most t honest parties. But still $\mathcal{C}^\beta$ is bound to contain at least $2t+1$ honest parties.

We now show that the above event happens after finite number of steps. We prove this by showing that an honest D may compute $\mathcal{O}(n^2)$ distinct (n,t)-stars in G_D. This is because D applies Find-STAR on $\overline{G_D}$ every time after an edge is added to G_D and there can be $\mathcal{O}(n^2)$ edges in G_D. Now $(\mathcal{C}^\beta, \mathcal{D}^\beta)$ with $\mathcal{C}^\beta$ containing at least $2t+1$ parties will be one among these $\mathcal{O}(n^2)$ (n,t)-stars. □

Lemma 11.29 *In protocol Pf-Ver-Agree, if D is honest, then eventually $CORE$ will be generated.*

Proof: By Lemma 11.28, the honest D will eventually generate an $(\mathcal{C}^\beta, \mathcal{D}^\beta)$ in G_D, with $\mathcal{C}^\beta$ containing at least $2t+1$ honest parties. Furthermore, if D is honest then eventually there will be edges between every pair of honest parties in the graph G_i of every honest P_i (including G_D). Thus, as all honest parties in $\mathcal{P}$ will have edges with the honest parties in $\mathcal{C}^\beta$, they will be eventually added to $\mathcal{F}^\beta$. Similarly, as all honest parties in $\mathcal{P}$ will have edges with the honest parties in $\mathcal{F}^\beta$, they will be eventually added to $\mathcal{E}^\beta$. Hence $|\mathcal{E}^\beta| \geq n-t$ and $|\mathcal{F}^\beta| \geq n-t$ will be satisfied and $CORE$ will be obtained by honest D. □

Lemma 11.30 *If an honest P_i has accepted $CORE$, then the row polynomials of the honest parties in $CORE$ define a unique bivariate polynomial of degree-(τ, t).*

Proof: If an honest P_i has accepted $CORE$, then he has received $((\mathcal{C}^\beta, \mathcal{D}^\beta), (\mathcal{E}^\beta, \mathcal{F}^\beta))$ from the A-cast of D and checked their validity with respect to his own graph G_i. By Lemma 11.27, the row polynomials of the honest parties in $\mathcal{C}^\beta$ define a unique bivariate polynomial of degree-(τ, t), say $\overline{F}(x,y)$. So the row polynomial held by an honest $P_i \in \mathcal{C}$ satisfies $\overline{f_i}(x) = \overline{F}(x,i)$. Now by the construction of $\mathcal{F}^\beta$, every *honest* $P_j \in \mathcal{F}^\beta$ has at least $2t+1$ neighbors in $\mathcal{C}^\beta$ which implies that $\overline{f_{kj}}$ values received from at least $2t+1$ parties in $\mathcal{C}^\beta$ lie on column polynomial $\overline{p_j}(y)$. This clearly implies $\overline{p_j}(y) = \overline{F}(j,y)$, as $t+1$ out of these $2t+1$ values are sent by honest parties in $\mathcal{C}$, who define $\overline{F}(j,y)$.

Similarly, by construction of $\mathcal{E}^\beta$, every *honest* $P_j \in \mathcal{E}^\beta$ has at least $\tau+t+1$ neighbors in $\mathcal{F}^\beta$ which implies that $\overline{p_{kj}}$ values received from at least $\tau+t+1$ parties in $\mathcal{F}^\beta$ lie on $\overline{f_j}(x)$. This implies that $\overline{f_j}(x) = \overline{F}(x,j)$, as at least $\tau+1$ out of these $\tau+t+1$ values are sent by honest parties in $\mathcal{F}^\beta$, who define $\overline{F}(x,j)$. Hence row polynomials of the honest parties in $CORE$ define $\overline{F}(x,y)$. □

11.6.3 Generation of τ-$(1d)$-sharing Phase

Assuming that the honest parties in $\mathcal{P}$ have agreed upon a $CORE$, protocol Pf-Gen generates τ-$(1d)$-sharing in the following way: From the properties of $CORE$, the row polynomials of honest parties in $CORE$ define a unique bivariate polynomial say $\overline{F}(x, y)$ of degree-(τ, t), such that each honest party P_i in $CORE$ possesses $\overline{f_i}(x) = \overline{F}(x, i)$. So the j^{th} point on $\overline{f_i}(x)$ polynomials corresponding to all honest P_i's in $CORE$, define degree-t polynomial $\overline{p_j}(y) = \overline{F}(j, y)$. Furthermore, $|CORE| \geq 3t + 1$. So the parties in $CORE$ can enable each $P_j \in \mathcal{P}$ to privately reconstruct $\overline{p_j}(y)$ using OEC [35] (also recall from Section 11.1.6). Once this is done, every P_j can output $\overline{p_j}(0)$ as the share of D's committed secret. Since $\overline{f_0}(j) = \overline{p_j}(0)$, it follows that $\overline{f_0}(0)(= \overline{F}(0, 0))$ will be τ-$(1d)$-sharing using the degree-τ polynomial $\overline{f_0}(x) = \overline{F}(x, 0)$. Clearly if D is honest, D's secret s will be τ-$(1d)$-sharing using polynomial $f_0(x) = F(x, 0)$, as $\overline{F}(x, y) = F(x, y)$ for honest D. The protocol is formally given in Fig. 11.14.

Figure 11.14: Third Phase of protocol Pf-AVSS-Share: **Generation of τ-$(1d)$-sharing**

Protocol Pf-Gen($D, \mathcal{P}, s, \tau$)

Code for P_i: Every party executes this code

1. Apply OEC technique [35] on $\overline{f_{ji}}$'s received from every P_j in $CORE$ (during Protocol Pf-Ver-Agree) and reconstruct degree-t polynomial $\overline{p_i}(y)$ and output $\overline{s_i} = \overline{p_i}(0) = \overline{f_0}(i)$ as the i^{th} share of $\overline{s}$ and terminate. $\overline{s}$ is now τ-$(1d)$-sharing using degree-τ polynomial $\overline{f_0}(x) = \overline{F}(x, 0)$.

Lemma 11.31 *Assume that every honest party has agreed on $CORE$ where the row polynomials of the honest parties in $CORE$ define a unique bivariate polynomial of degree-(τ, t), say $\overline{F}(x, y)$. Then protocol P-Gen will generate τ-sharing of $\overline{s} = \overline{F}(0, 0)$.*

PROOF: To achieve τ-$(1d)$-sharing of $\overline{s}$ using polynomial $\overline{f_0}(x)$, party P_i should hold $\overline{f_0}(i)$ as i^{th} share of $\overline{s}$. Now $\overline{f_0}(i) = \overline{p_i}(0)$ holds by the property of bivariate polynomial. Also by property of $CORE$, the i^{th} point on $\overline{f_j}(x)$ polynomials, corresponding to honest P_j's in $CORE$ define degree-t polynomial $\overline{p_j}(y)$. So P_i can apply OEC on $\overline{f_{ji}}$'s received from the parties in $CORE$ (during Protocol Pf-Ver-Agree), reconstruct $\overline{p_j}(y)$ and obtain $\overline{p_j}(0)$ which is i^{th} share of $\overline{s}$. □

11.6.4 Protocol Pf-AVSS-Share and Pf-AVSS-Rec

The protocol for our perfect AVSS scheme is given in Fig. 11.15.

Theorem 11.32 *Protocol Pf-AVSS consisting of sub-protocols (Pf-AVSS-Share, Pf-AVSS-Rec) constitute a valid perfect AVSS scheme for sharing a single secret from $\mathbb{F}$ (according to Definition 11.1).*

PROOF: **Termination:** Part (1) of **Termination** says that if D is honest then every honest party will terminate Pf-AVSS-Share eventually. By Lemma 11.29, D will eventually generate $CORE$ and A-cast the corresponding information i.e. $((\mathcal{C}^\beta, \mathcal{D}^\beta), (\mathcal{E}^\beta, \mathcal{F}^\beta))$. By the property of A-cast (and as graph G_i is constructed on the basis of A-casted information) every honest party will receive, verify the

Figure 11.15: Perfect AVSS protocol: Pf-AVSS

Protocol Pf-AVSS-Share($D, \mathcal{P}, s, \tau$)

1. D executes Pf-Distr($D, \mathcal{P}, s, \tau$);
2. Each party P_i participates in Pf-Ver-Agree($D, \mathcal{P}, s, \tau$);
3. After agreeing on $CORE$, each party P_i participates in Pf-Gen($D, \mathcal{P}, s, \tau$) and terminates Pf-AVSS-Share after locally outputting the share corresponding to D's committed secret.

Protocol Pf-AVSS-Rec($D, \mathcal{P}, s, \tau$)

Code for P_i:

1. Privately send s_i, the i^{th} share of the secret to every $P_j \in \mathcal{P}$.
2. Apply OEC on received s_j's, reconstruct the secret and terminate Pf-AVSS-Rec.

validity of D's A-casted information with respect to his own graph G_i and agree on the $CORE$. Now the proof for this part follows from Lemma 11.31.

Part (2) of **Termination** says that if an honest party terminated Pf-AVSS-Share, then every other honest party will terminate Pf-AVSS-Share eventually. An honest P_i has terminated the protocol implies that he has agreed on $CORE$. This means that P_i has received and verified the validity of D's A-casted information with respect to his own graph G_i. The same will happen eventually for all other honest parties. Hence they will agree on $CORE$. Now the proof follows from Lemma 11.31.

Part (3) of **Termination** follows from the properties of $CORE$ and OEC.

Correctness: If the honest parties terminate Pf-AVSS-Share, then it implies that $\overline{s}(=\overline{F}(0,0))$ is properly τ-($1d$)-sharing among the parties in $\mathcal{P}$ (by Lemma 11.31), where $\overline{F}(x,y)$ is the unique bivariate polynomial of degree-(τ, t) defined by the row polynomials of the honest parties in $CORE$. Moreover if D is honest then $\overline{F}(x,y) = F(x,y)$ (follows from Lemma 11.27 and Lemma 11.30) and hence $\overline{s} = s$. Now the **Correctness** follows from the correctness of OEC.

Secrecy: Let $\mathcal{A}_t$ controls $P_1, \ldots, P_t$. So $\mathcal{A}_t$ will know $f_1(x), \ldots, f_t(x)$ and $p_1(y), \ldots, p_t(y)$. Throughout the protocol, the parties exchange common values (on row and column polynomials), which do not add any extra information to the view of $\mathcal{A}_t$. Now by the property of bivariate polynomial of degree-(τ, t), $\tau - t + 1$ coefficients of $f_0(x) = F(x, 0)$ will remain secure, where $F(x, y)$ is the polynomial used by D to hide his secret s. So $s = f_0(0) = F(0, 0)$ will remain secure. $\square$

Theorem 11.33

- *Pf-AVSS-Share privately communicates $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits and A-casts $\mathcal{O}(n^2 \log n)$ bits.*
- *Protocol Pf-AVSS-Rec privately communicates $\mathcal{O}(n^2 \log |\mathbb{F}|)$.*

PROOF: In Pf-Distr, D privately communicates $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits. In Pf-Ver-Agree, the parties privately communicate $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits. In addition, the parties also A-cast $\mathtt{OK}(.,.)$s, which requires A-cast communication of $\mathcal{O}(n^2 \log n)$ bits (each $\mathtt{OK}(.,.)$ signal can be represented by $\mathcal{O}(\log n)$ bits, as the signal contains identity of two parties and the identify of any party from the set of n parties $\mathcal{P}$ can be represented by $\log n$ bits). Furthermore, A-casting $((\mathcal{C}^\beta, \mathcal{D}^\beta), (\mathcal{E}^\beta, \mathcal{F}^\beta)))$ by D requires A-casting of $\mathcal{O}(n \log n)$ bits (the identify of a party can be represented by $\log n$ bits, as there are n different parties). In Pf-Gen, no communication is performed. So in total, Pf-AVSS-Share requires private communication of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits and A-cast of $\mathcal{O}(n^2)$ bits.

In Pf-AVSS-Rec, the parties in $\mathcal{P}$ send their shares to every party in $\mathcal{P}$. So Pf-AVSS-Rec requires $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits of private communication. □

11.7 Perfect AVSS for Sharing Multiple Secrets

We now present a perfect AVSS scheme, called Pf-AVSS-MS, consisting of pair of protocols, namely (Pf-AVSS-MS-Share, Pf-AVSS-MS-Rec): Pf-AVSS-MS-Share allows a dealer $D \in \mathcal{P}$ to τ-$(1d)$-share $\ell \geq 1$ secret(s) from $\mathbb{F}$, denoted as $S = (s^1, \ldots, s^\ell)$, among the parties in $\mathcal{P}$, where $t \leq \tau \leq 2t$; Pf-AVSS-MS-Rec allows the parties to reconstruct the secrets, given their τ-$(1d)$-sharing. Notice that we can generate τ-$(1d)$-sharing of S by concurrently executing protocol Pf-AVSS-Share (given in the previous section) ℓ times, once for each $s^i \in S$. But this will require a private communication of $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ and A-cast of $\mathcal{O}(\ell n^2)$ bits. However, our protocol Pf-AVSS-MS-Share requires a private communication of $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ and A-cast of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits. *Thus, the A-cast communication of our Pf-AVSS-MS-Share protocol is independent of ℓ.* The idea behind protocol Pf-AVSS-MS-Share is same as Pf-AVSS-Share. Protocol Pf-AVSS-MS-Share is divided into a sequence of same three phases, as in Pf-AVSS-Share. We now present the corresponding protocols in Fig. 11.16, Fig. 11.17 and Fig. 11.18.

Figure 11.16: First Phase of Protocol Pf-AVSS-MS-Share: **Distribution by** D Phase

Protocol Pf-Distr-MS($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \tau$**)**

Code for D**:** Only D executes this code

1. For $l = 1, \ldots, \ell$, select a random bivariate polynomials $F^l(x,y)$ of degree-(τ, t), such that $F^l(0,0) = s^l$ and send the row polynomial $f_i^l(x) = F^l(x,i)$ and column polynomial $p_i^l(y) = F^l(i,y)$ to party P_i, for $i = 1, \ldots, n$.

Remark 11.34 *In protocol Pf-Ver-Agree-MS, in step i.(4), instead of A-casting ℓ $\mathtt{OK}(P_i, P_j)$ signals, party P_i A-casts a single $\mathtt{OK}(P_i, P_j)$ signal after verifying the consistency of common values on all the ℓ row and column polynomials. It is this step, which makes the A-cast communication of Pf-Ver-Agree-MS, independent of ℓ. A similar idea is also used in the AVSS scheme of [13], which generates t-$(1d)$-sharing of ℓ secrets concurrently.*

Figure 11.17: Second Phase of Protocol Pf-AVSS-MS-Share: **Verification & Agreement on CORE** phase

Protocol Pf-Ver-Agree-MS($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \tau$)

i. Code for P_i: Every party $P_i \in \mathcal{P}$ (including D) executes this code.

1. Wait to receive $\overline{f_i^l}(x)$ and $\overline{p_i^l}(y)$ for all $l = 1, \ldots, \ell$, from D.
2. Upon receiving, check whether (i) $\overline{f_i^l}(x)$ is a degree-τ polynomial for all $l = 1, \ldots, \ell$; and (ii) $\overline{p_i^l}(y)$ is a degree-t polynomial for all $l = 1, \ldots, \ell$. If yes, then send $\overline{f_{ij}^l} = \overline{f_i^l}(j)$ and $\overline{p_{ij}^l} = \overline{p_i^l}(j)$ for all $l = 1, \ldots, \ell$, to P_j.
3. Upon receiving $\overline{f_{ji}^1}, \ldots, \overline{f_{ji}^\ell}$ and $\overline{p_{ji}^1}, \ldots, \overline{p_{ji}^\ell}$ from P_j, check if $\overline{f_i^l}(j) \stackrel{?}{=} \overline{p_{ji}^l}$ and $\overline{f_{ji}^l} \stackrel{?}{=} \overline{p_i^l}(j)$ for all $l = 1, \ldots, \ell$. If the equality holds, then confirm the consistency by A-casting $\texttt{OK}(P_i, P_j)$.
4. Construct an undirected graph G_i with $\mathcal{P}$ as vertex set. Add an edge (P_j, P_k) in G_i upon receiving (a) $\texttt{OK}(P_k, P_j)$ from the A-cast of P_k and (b) $\texttt{OK}(P_j, P_k)$ from the A-cast of P_j.

ii. Code for D: (Only D executes this code): Same as in Protocol Pf-Ver-Agree.

iii. Code for P_i: (Every party $P_i \in \mathcal{P}$ (including D) executes this code): Same as in Protocol Pf-Ver-Agree.

Figure 11.18: Third Phase of protocol Pf-AVSS-MS-Share: **Generation of τ-($1d$)-sharing** Phase

Protocol Pf-Gen-MS($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \tau$)

Code for P_i: Every party executes this code

1. For $l = 1, \ldots, \ell$, apply OEC technique on $\overline{f_{ji}^l}$'s received from every P_j in $CORE$ (during Protocol Pf-Ver-Agree) and reconstruct degree-t polynomial $\overline{p_i^l}(y)$ and output $\overline{s_i^l} = \overline{p_i^l}(0) = \overline{f_0^l}(i)$ as the i^{th} share of $\overline{s^l}$ and terminate. $\overline{s^l}$ is now τ-($1d$)-sharing using degree-τ polynomial $\overline{f_0^l}(x)$.

Protocol Pf-AVSS-MS-Share and Pf-AVSS-MS-Rec are now given in the Fig. 11.19.

Theorem 11.35 *Protocol Pf-AVSS-MS consisting sub-protocols (Pf-AVSS-MS-Share, Pf-AVSS-MS-Rec) constitutes a valid perfectly secure AVSS scheme, which concurrently shares $\ell \geq 1$ elements from $\mathbb{F}$ (according to Definition 11.1).*

Theorem 11.36 (Communication Complexity of Pf-AVSS-MS)

Figure 11.19: Our Perfect AVSS protocol: Protocol Pf-AVSS-MS

Protocol Pf-AVSS-MS-Share($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \tau$)

1. D executes Pf-Distr-MS($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \tau$);
2. Each party P_i participates in Pf-Ver-Agree-MS($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \tau$);
3. After agreeing on $CORE$, each party P_i participates in Pf-Gen-MS($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \tau$) and terminates Pf-AVSS-MS-Share after locally outputting the shares corresponding to D's committed secrets.

Protocol Pf-AVSS-MS-Rec($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \tau$)

1. For $l = 1, \ldots, \ell$, each party $P_i \in \mathcal{P}$ privately sends the i^{th} share of s^l, namely s^l_i, to every party $P_j \in \mathcal{P}$.
2. For $l = 1, \ldots, \ell$, party $P_i \in \mathcal{P}$ applies OEC on the received s^l_j's to privately reconstruct s^l and terminate Pf-AVSS-MS-Rec.

- *Protocol Pf-AVSS-MS-Share privately communicates $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits and A-casts $\mathcal{O}(n^2 \log n)$ bits.*
- *Protocol PAVSS-MS-Rec privately communicates $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits.*

PROOF: In Pf-Distr-MS, D privately communicates $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits. In Pf-Ver-Agree, the parties privately communicate $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits. In addition, the parties also A-cast OK(.,.)s, which requires A-cast communication of $\mathcal{O}(n^2 \log n)$ bits (each OK(.,.) signal can be represented by $\mathcal{O}(\log n)$ bits, as the signal contains identity of two parties and the identify of any party from the set of n parties $\mathcal{P}$ can be represented by $\log n$ bits). Furthermore, A-casting $((\mathcal{C}^\beta, \mathcal{D}^\beta), (\mathcal{E}^\beta, \mathcal{F}^\beta)))$ by D requires A-casting of $\mathcal{O}(n \log n)$ bits. In Pf-Gen-MS, no communication is performed. So in total, Pf-AVSS-MS-Share requires private communication of $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits and A-cast of $\mathcal{O}(n^2)$ bits.

In Pf-AVSS-MS-Rec, the parties in $\mathcal{P}$ send their shares to every party in $\mathcal{P}$. So Pf-AVSS-MS-Rec requires $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits of private communication. □

11.8 A Different Interpretation of Protocol Pf-AVSS-MS

In Pf-AVSS-MS-Share, every secret s^l for $l = 1, \ldots, \ell$ is τ-(1d)-sharing using degree-τ polynomial $f^l_0(x) = F^l(x, 0)$, where $t \leq \tau \leq 2t$. Now by the **Secrecy** proof of Pf-AVSS-Share, given in Theorem 11.32, we can claim that $(\tau + 1) - t$ coefficients of $f^l_0(x)$ are information theoretically secure for every $l = 1, \ldots, \ell$. This implies that Pf-AVSS-MS-Share shares $\ell(\tau + 1 - t)$ secrets with a private communication of $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits and A-cast $\mathcal{O}(n^2 \log n)$ bits. As the A-cast communication is independent of ℓ, we may ignore it and conclude that the amortized cost of sharing a single secret using Pf-AVSS-MS-Share is only $\mathcal{O}(n \log |\mathbb{F}|)$. This is because by setting $\tau = 2t$ (the maximum value of τ), we see that Pf-AVSS-MS-Share can share $\ell(t + 1) = \Theta(\ell n)$ secrets by privately communicating

$\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits. Now putting it in other way, D can share $\ell(t+1)$ secrets using Pf-AVSS-MS-Share by choosing a random polynomial $f_0^l(x)$ (of degree $\tau = 2t$) with lower order $t+1$ coefficients as secrets and then choosing a random degree-(τ, t) bivariate polynomial $F^l(x, y)$ with $F^l(x, 0) = f_0^l(x)$ for $l = 1, \ldots, \ell$ and finally executing Pf-AVSS-MS-Share with $F^1(x, y), \ldots, F^\ell(x, y)$.

A similar interpretation holds for protocol St-AVSS-MS-Share as well (as presented in Section 11.4). However, recall that protocol St-AVSS-MS-Share generates τ-sharing of ℓ secrets with high probability and hence may involve a negligible error probability. On the other hand protocol Pf-AVSS-MS-Share is perfect in all respect and does not involve any error probability.

Finally, we now mention another application of Pf-AVSS-MS-Share which uses the above interpretation. Using protocol Pf-AVSS-MS-Share, we can design an ABA protocol with an amortized communication cost of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits for reaching agreement on a single bit. To the best of our knowledge, there is only one ABA with $4t+1$ due to [66] which requires fairly high communication complexity (though polynomial in n). We will elaborate on our ABA in Chapter 13.

Remark 11.37 *The best known perfect AVSS of [13] requires an amortized cost $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits for sharing a single secret. Hence Pf-AVSS-MS-Share shows a clear improvement over the AVSS of [13].*

Remark 11.38 *The idea of hiding multiple secrets in a single polynomial was explored earlier in [83] in the context of passive adversary in synchronous network. Doing the same in asynchronous network, in the presence of active adversary is bit tricky and calls for new techniques. Though we can hide $(\tau+1-t)$ secrets in each degree-τ polynomial $f_0^l(x)$ using protocol Pf-AVSS-MS-Share, we will hide only one secret, namely s^l in $f_0^l(x)$. This is because in our AMPC protocol (presented in the next chapter), we require that each degree-τ polynomial hides only one secret. However, hiding multiple secrets in a degree-τ polynomial will be useful in the context of ABA, presented in Chapter 13.*

11.9 Conclusion and Open Problems

In this chapter, we designed two AVSS protocols with $4t+1$ parties in which one is statistical (and thus have non-optimal resilience) and the other one is perfect, along with being optimally resilient. Both our AVSS protocols are based on completely disjoint techniques. Yet, both our AVSSs are capable of generating τ-$(1d)$-sharing of secret(s) for any $t \leq \tau \leq 2t$. When we have $n = 4t+1$ parties, τ-$(1d)$-sharing tremendously simplifies the computation of multiplication gate in an AMPC protocol. In the next chapter, the statistical AVSS and the perfect AVSS are used for constructing our statistical AMPC and perfect AMPC with $n = 4t+1$. There we show how our AVSS protocols simplify computation of a multiplication gate.

We conclude this chapter which the following open questions:

Open Problem 18 *Can we design statistical AVSS protocol (with non-optimal resilience) with better communication complexity than what is reported here?*

Open Problem 19 *Can we design perfect AVSS protocol (with optimal resilience) with better communication complexity than what is reported here?*

Chapter 12

Efficient Statistical AMPC Protocol With Non-Optimal Resilience and Perfect AMPC With Optimal Resilience

AMPC without any error in computation (also called as perfect AMPC) is possible iff $n \geq 4t + 1$. When a negligible error probability of ϵ is allowed in the computation, then the AMPC is called as statistical AMPC. Statistical AMPC is possible iff $n \geq 3t + 1$. In this chapter, our focus is on AMPC designed with $n = 4t + 1$ parties, both with and without error in computation. Precisely, we focus on the communication complexity of the AMPC protocols with $n = 4t + 1$ parties.

Communication complexity, being one of the important parameters of AMPC protocol, drew quite a bit of attention and hence there are a number of attempts to improve the communication complexity of AMPC protocols (both with error and without error) with $4t + 1$ parties. The latest such attempt is reported in [107] where the authors presented a statistical AMPC protocol with $n = 4t + 1$ that communicates $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits per *multiplication gate*, where $\mathbb{F}$ is the finite field over which the computation of the protocol is carried out. However, in this chapter we show that the protocol of [107] is not a correct statistical AMPC. We then present a new, simple, statistical AMPC protocol with $n = 4t + 1$ which communicates $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits per *multiplication gate*. Moving a step forward, we also present a perfect AMPC protocol which communicates $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits per *multiplication gate*. Now it is important to note that not only our perfect AMPC protocol is able to achieve the same communication complexity as our statistical AMPC protocol, but also it is now optimally resilient (that is, it is designed with $n = 4t + 1$ parties) which is not the case in our statistical AMPC protocol. The best known perfect AMPC protocol with optimal resilience [13] communicates $\mathcal{O}(n^3 \log |\mathbb{F}|)$ bits per multiplication gate. Hence our AMPC protocol provides the best communication complexity among all the known AMPC protocols.

As a key tool for our statistical AMPC and perfect AMPC, we use our statistical AVSS and perfect AVSS respectively, presented in Chapter 11. In this chapter, we reveal how τ-$(1d)$-sharing of secrets (that can be generated by our AVSS protocols) simplifies the computation of multiplication gate of AMPC protocol, compared to the computation done for the same in AMPC protocol with

$n = 3t + 1$ parties presented in Chapter 10.

12.1 Introduction

12.1.1 The Network and Adversary Model

This is same as described in section 8.1.1. Here we recall that the set of parties is denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$ and t out of the n parties can be under the influence of a *computationally unbounded Byzantine (active) adversary*, denoted as $\mathcal{A}_t$. We emphasize that we use $n = 4t + 1$ in this chapter.

12.1.2 Definitions

Using our AVSS protocols as black box, we design another important protocol to generate $(t, 2t)$-$(1d)$-sharing of secrets which will be used to evaluate multiplication gate of a circuit. The $(t, 2t)$-$(1d)$-sharing of a secret s is defined as follows:

Definition 12.1 ($(t, 2t)$-$(1d)$-sharing) *A value s is said to be $(t, 2t)$-$(1d)$-shared among the parties in $\mathcal{P}$, denoted as $[s]_{(t,2t)}$, if s is both t-$(1d)$-shared and $2t$-$(1d)$-shared.*

12.1.3 Relevant Literature on AMPC

Unlike MPC in synchronous networks, designing AMPC protocols has received less attention due to their inherent difficulty. Since in this chapter, our focus is on information theoretic security (that is achieved against adversary having *unbounded computing power*), we channelize our focus mainly on AMPC protocols that provides information theoretic security. Such AMPC protocols can be categorized mainly into two types:

1. *Perfect AMPC:* An AMPC protocol which satisfies all the three properties, namely, **correctness, secrecy** and **termination** without any error is called *perfect* AMPC. In [19], it is shown that perfect AMPC is possible iff $n \geq 4t + 1$. Thus any perfect AMPC designed with $n = 4t + 1$ is said to be *optimally resilient*. Optimally resilient, perfect AMPC protocols are reported in [19, 143, 13]. Among these, the AMPC protocol of [13] provides the best communication complexity, which is $\mathcal{O}(n^3 \log(|\mathbb{F}|)) = \mathcal{O}(n^3 \log n)$ bits *per multiplication gate*, where $|\mathbb{F}| \geq n$.

2. *Statistical AMPC:* An AMPC protocol which satisfies **correctness** AND/OR **termination** condition except with negligible error probability of ϵ is called *statistical* AMPC. However, notice that there is no compromise in **secrecy** property. From [21], it is known that statistical AMPC is possible iff $n \geq 3t + 1$. Thus any statistical AMPC protocol designed with $n = 3t + 1$ is said to be *optimally resilient*. Optimally resilient, statistical AMPC are reported in only [21] and in this thesis in Chapter 10. Among these, the AMPC protocol of this thesis provides the better communication complexity which is $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits *per multiplication gate*, where the protocol works over a field $\mathbb{F} = GF(2^\kappa)$ and each element of $\mathbb{F}$ is represented by $\kappa = \mathcal{O}(\log \frac{1}{\epsilon})$ bits.

In comparison to perfect AMPC, statistical AMPC protocols reported in the literature have much more communication complexity. To achieve better communication complexity for the statistical AMPC protocols, researchers have tried to design statistical AMPC with non-optimal resilience i.e with $n = 4t + 1$ parties. Such AMPC protocols are reported in [135] and recently in [107][1]. While the AMPC of [135] achieves a communication complexity of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits per multiplication gate, the AMPC of [107] claims to achieve communication complexity of $\mathcal{O}(n^2 \log \frac{1}{\epsilon})$ bits per multiplication gate (in this chapter we show that AMPC of [107] is not a correct statistical AMPC protocol). Both the AMPC protocols of [135] and [107] are based on *player elimination* framework of [98], an important technique introduced in synchronous network in order to reduce communication complexity of MPC protocols.

The communication complexity (per multiplication gate) of known AMPC protocols and the AMPC protocol presented in this thesis so far, is summarized in Table 12.1.

Table 12.1: Communication complexity (CC) in bits per multiplication gate of known AMPC protocols.

Reference	Type	Resilience	CC in bits
[19, 35]	Perfect	$t < n/4$ (optimal)	$\mathcal{O}(n^6 \log n)$
[143]	Perfect	$t < n/4$ (optimal)	$\Omega(n^5 \log n)$
[13]	Perfect	$t < n/4$ (optimal)	$\mathcal{O}(n^3 \log n)$
[21]	Statistical	$t < n/3$ (optimal)	$\Omega(n^{11} (\log \frac{1}{\epsilon})^4)$
Chapter 10	Statistical	$t < n/3$ (optimal)	$\mathcal{O}(n^5 \log \frac{1}{\epsilon})$
[135]	Statistical	$t < n/4$ (non-optimal)	$\mathcal{O}(n^4 \log \frac{1}{\epsilon})$
[107]	Statistical	$t < n/4$ (non-optimal)	$\mathcal{O}(n^2 \log \frac{1}{\epsilon})$

AMPC under *cryptographic assumptions* is possible iff $n \geq 3t + 1$ [105, 106]. The best known AMPC under cryptographic assumptions is due to [106], which communicates $\mathcal{O}(n^2 \kappa)$ bits per multiplication gate, where κ is the security parameter.

Recently in [15], the authors have designed communication efficient MPC protocols over networks that exhibit partial asynchrony (where the network is synchronous up to certain point and becomes completely asynchronous after that). In another work, Damgård et al. [51] have reported efficient MPC protocol over a network that assumes the concept of synchronization point; i.e,. the network is asynchronous before and after the synchronization point. We will not consider the protocols of [15] and [51] for further discussion as they are not designed in completely asynchronous settings in which our AMPC protocols are proposed.

[1]Though it is not explicitly stated in [107], the AMPC protocol of [107] involves error probability in **termination** and **correctness**

12.1.4 Contribution of This Chapter

In this chapter our focus is on AMPC with $4t+1$ parties. Our main contributions for AMPC are:

1. From Table 12.1, we find that the most communication efficient, statistical AMPC protocol is due to [107]. However, we show that this protocol does not satisfy the termination and correctness properties of statistical AMPC.
2. We then design a new statistical AMPC protocol with $n = 4t + 1$, which communicates $\mathcal{O}(n^2 \log n)$ bits per multiplication gate. Our protocol is simple and achieves its goal *without* using *player elimination* framework of [98] (which is used in [107]).
3. Finally we present a new, perfect AMPC protocol with $n = 4t + 1$ which communicates $\mathcal{O}(n^2 \log n)$ bits per *multiplication gate*. Now it is important to note that not only our perfect AMPC is able to achieve the same communication complexity as our statistical AMPC, but it is now optimally resilient. From Table 12.1, the best known perfect AMPC with optimal resilience [13] communicates $\mathcal{O}(n^3 \log n)$ bits per multiplication. Hence our AMPC protocol provides the best communication complexity among all the known AMPC protocols.

For designing our AMPC protocols, we use our statistical and perfect AVSS protocols presented in Chapter 11.

Our protocols for perfect AVSS and AMPC work on a field $\mathbb{F}$ with $|\mathbb{F}| \geq n$. Hence every element from $\mathbb{F}$ can be represented by $\log |\mathbb{F}| = \mathcal{O}(\log n)$ bits. On the other hand, for statistical AVSS and AMPC, we use two fields called ground field and extension field, which are defined as follows:

The Ground Field and The Extension Field: The field $\mathbb{F}$ that is used in perfect AVSS and AMPC is denoted as *ground field*. Most of the computation of statistical AVSS and AMPC is performed over this field. We also fix an extension field $\mathbb{E} \supset \mathbb{F}$ to be smallest extension for which $|\mathbb{E}| \geq 2^\kappa = \frac{1}{\epsilon}$, where ϵ is the error parameter. Each element of $\mathbb{E}$ can be represented using $\mathcal{O}(\kappa) = \mathcal{O}(\log \frac{1}{\epsilon})$ bits. We call $\mathbb{E}$ as *Extension Field*. Moreover, without loss of generality, we assume that $n = \text{poly}(\kappa)$. Some of the computation of our statistical AVSS and AMPC is performed over $\mathbb{E}$ so as to bound the error probability of the protocols by ϵ.

12.1.5 Primitives Used

In this chapter, we require $\mathsf{A\text{-}cast}$ (recalled in Chapter 7) and Online Error Correction (OEC) technique (recalled in Chapter 11). Apart from these, we require ACS in our AMPC protocols. In the sequel we recall ACS protocol (it was very briefly discussed in Section 10.3 of Chapter 10). A deeper understanding of the protocol is necessary to understand the fault in statistical AMPC of Huang et al. [107].

12.1.5.1 Agreement on a Common Subset (ACS)

ACS is an asynchronous primitive presented in [19, 21], which allows all (honest) parties in $\mathcal{P}$ to agree on a common set of at least $n-t$ parties, who will eventually satisfy some property, say Q, where Q has the following characteristics:

1. It is known that every honest party will eventually satisfy Q.
2. Some corrupted parties may also satisfy Q.
3. If some *honest* $P_j \in \mathcal{P}$ knows that some party $P_\alpha \in \mathcal{P}$, satisfies Q, then every other *honest* party in $\mathcal{P}$ will also eventually conclude that P_α satisfies Q.

For example, consider the following scenario: suppose that all the parties in $\mathcal{P}$ are asked to A-cast some value(s) and the property Q is whether a party has A-casted the value(s) or not. It is easy to see that Q satisfies all the above three characteristics. This is because (a) Every honest party will eventually A-cast the value(s); (b) Some corrupted parties may also A-cast the value(s); (c) If some *honest* $P_j \in \mathcal{P}$ knows that some $P_\alpha \in \mathcal{P}$ satisfies Q, then it implies that P_j has received the value(s) A-casted by P_α. So by the property of A-cast, every other *honest* party in $\mathcal{P}$ will also eventually receive those values from P_α. In short, by using ACS primitive, the (honest) parties can eventually agree on a set of $n-t$ parties who have broadcasted some value(s).

Another example of property Q could be that a party has AVSS-shared some value(s). The idea behind ACS protocol is to execute n instances of ABA [35], one on behalf of each party, to decide whether it will be in the common set. For the sake of completeness, we present the protocol in Fig. 12.1. The current description of protocol ACS is taken from [21].

Figure 12.1: Protocol for Agreement on a Common Subset with $n = 4t + 1$

Protocol ACS

Code for Party P_i: Every party in $\mathcal{P}$ executes this code

1. For each $P_j \in \mathcal{P}$ for whom you know that $Q(j) = 1$ (i.e., P_j satisfies property Q), participate in ABA_j with input 1. Here for $j = 1, \ldots, n$, ABA_j denotes the instance of *asynchronous Byzantine Agreement* (ABA) executed with respect to $P_j \in \mathcal{P}$ to decide whether P_j will be in the common set.
2. Upon terminating $n-t$ instances of ABA with output 1, enter input 0 to all other instances of ABA, for which you haven't entered a value yet.
3. Upon terminating all the n ABA protocols, let your $SubSet_i$ be the set of all indices j for which ABA_j had output 1.
4. Output the set of parties corresponding to the indices in $SubSet_i$ and terminate ACS.

Theorem 12.2 ([21]) *Using protocol ACS, the (honest) parties in $\mathcal{P}$ can agree on a common subset of at least $n-t$ parties, who will eventually satisfy property Q. The communication complexity of the protocol is $\mathcal{O}(poly(n))$.*

12.1.6 The Road-map

This chapter is organized as follows: Section 12.2 shows that AMPC protocol presented in [107] is not a correct statistical AMPC protocol. Section 12.3 presents a statistical protocol for generating $(t, 2t)$-$(1d)$-sharing of multiple secrets. Section 12.4 describes our statistical AMPC protocol. Next, section 12.5 presents a perfect protocol for generating $(t, 2t)$-$(1d)$-sharing of multiple secrets and section 12.6 presents our perfect AMPC protocol. Lastly, we conclude this chapter in section 12.7.

12.2 Statistical AMPC of Huang et al. [107]

We now recall the statistical AMPC protocol of [107] and show that it does not satisfy the **correctness** and **termination** property of AMPC. The AMPC protocol of [107] is based on pre-processing model of [5]. Specifically, the AMPC protocol of [107] is divided into a sequence of three phases, namely **Pre-computation Phase** or **Preparation Phase**, **Input Phase** and **Computation & Output Phase**. We concentrate on **Preparation Phase** and show that it fails to satisfy its correctness and termination property, as claimed in [107]. This will further imply that the AMPC of [107] does not satisfy correctness and termination property.

The goal of the **Pre-Computation Phase** is to generate c_M random multiplication triples $(a^1, b^1, c^1), \ldots, (a^{c_M}, b^{c_M}, c^{c_M})$, where for $i = 1, \ldots, c_M$, each a^i, b^i and c^i are t-$(1d)$-shared among the parties in $\mathcal{P}$ with a^i and b^i being random and c^i satisfying $c^i = a^i \cdot b^i$. For this, the authors used *batch secret sharing scheme* (BSS) from [152]. In [152], the authors claimed that their BSS protocol correctly generates c_M random multiplication triples over $\mathbb{F}$. Moreover, every honest party will eventually terminate BSS. However, we will now show that their BSS scheme does not satisfy any of these two properties. As a result, the AMPC protocol of [107] (which uses the BSS scheme as a black box) does not satisfy **correctness** and **termination**.

The BSS scheme of [152] is based on *player elimination* framework [98], where the computation is divided into a sequence of segments. In order to show the weakness in the BSS scheme of [152], we need not have to go into the details of the player elimination framework. We concentrate only on the crucial steps (presented in a simplified form for the ease of presentation) which are executed in a segment to generate one t-$(1d)$-shared random multiplication triple (a, b, c). The two main steps in the generation of such triple are as follows:

1. The parties in $\mathcal{P}$ jointly generate t-$(1d)$-sharing of random values a and b.

2. The parties in $\mathcal{P}$ jointly compute t-$(1d)$-sharing of $c = ab$.

The t-$(1d)$-sharing of a and b in the BSS scheme of [152] is generated by executing the steps (presented in a simplified form for ease of presentation) presented in Fig. 12.2.

From Fig. 12.2, we find that step (2) to check whether a and b are indeed t-$(1d)$-shared among the parties in $\mathcal{P}$ will work if every (honest) $P_i \in \mathcal{P}$ holds a_i and b_i eventually. Clearly, this is possible if *every* (honest) party $P_i \in \mathcal{P}$ eventually receives $f_j(i)$ and $g_j(i)$ from *every* $P_j \in \mathcal{C}$. In [152], the authors claimed that by executing step (1) in Fig. 12.2, every (honest) $P_i \in \mathcal{P}$ will eventually receive

Figure 12.2: Steps for Generating t-$(1d)$-sharing of Random a and b in a Segment in the BSS Scheme of Zheng et al. [152]

1. **Generation of t-$(1d)$-sharing of a and b**: Code for Party $P_i \in \mathcal{P}$:

   (a) Select two random degree-t polynomials $f_i(x)$ and $g_i(x)$ and send $f_i(j), g_i(j)$ to every $P_j \in \mathcal{P}$. After sending, A-cast 1 to indicate that you have finished the sharing.

   (b) Participate in ACS protocol and input 1 in ABA_j (in ACS) *if you have received* 1 *from the* A-cast *of* P_j *AND if you have privately received* $f_j(i), g_j(i)$ *from* P_j.

   (c) Let $\mathcal{C}$ be the common set which is output by the ACS protocol, where $|\mathcal{C}| \geq n - t$.

   (d) Compute $a_i = \sum_{P_j \in \mathcal{C}} f_j(i)$ and $b_i = \sum_{P_j \in \mathcal{C}} g_j(i)$, as i^{th} share of a and b.

2. **Verifying whether indeed a and b are t-$(1d)$-shared among the parties in $\mathcal{P}$**: Here the parties perform some computation to check whether a and b are indeed shared using degree-t polynomials. If it is not the case then the segment fails and parties execute another protocol for *fault localization* (for details see [152]). However, the verification is carried out under the assumption that *every* (honest) party $P_i \in \mathcal{P}$ will eventually possess the share a_i and b_i of a and b respectively. For details, see [152].

$f_j(i)$ and $g_j(i)$ from *every* $P_j \in \mathcal{C}$ and hence will be able to compute a_i and b_i. However, we now show that $\mathcal{A}_t$ may behave in such a way that every honest P_i may wait indefinitely to compute a_i and b_i.

Without loss of generality, let the first $n-t$ parties in $\mathcal{P}$ (i.e $P_1, \ldots, P_{n-t}$) be honest and last t parties in $\mathcal{P}$ be corrupted. Now consider the following behavior of a *corrupted* $P_j \in \mathcal{P}$: P_j selects $f_j(x)$ and $g_j(x)$ of degree *more than* t and gives points on $f_j(x), g_j(x)$ to only *first* $n - 2t$ *honest parties and to the* t *corrupted parties* (but **not** to remaining t honest parties in $\mathcal{P}$). But still P_j A-casts 1 to indicate that he has sent the points to every party in $\mathcal{P}$. Moreover, $\mathcal{A}_t$ schedules the messages of P_j such that they reach to their respective receivers immediately, without any delay. Now $n-2t$ honest parties and t corrupted parties will input 1 (assuming the corrupted parties are behaving properly) in ABA_j in ACS, as they will receive points on $f_j(x)$ and $g_j(x)$ from P_j AND will also receive 1 from the A-cast of P_j. So in ABA_j, there are $n-t$ inputs, with value 1. Now assuming that all the parties including the corrupted parties behave properly in ABA_j, the property of ABA ensures that every party in $\mathcal{P}$ will terminate ABA_j with output 1 and hence P_j will be present in the common set $\mathcal{C}$. However, notice that the last t honest parties (to whom P_j has not sent the points on $f_j(x)$ and $g_j(x)$) did not feed any input in ABA_j. In fact, these honest parties will never receive their respective points on $f_j(x)$ and $g_j(x)$, in spite of terminating ABA_j with output 1. So even though a (corrupted) P_j is present in $\mathcal{C}$, potentially t honest parties may never receive their respective points on $f_j(x)$ and $g_j(x)$.

Now using similar strategy, another corrupted $P_k \in \mathcal{C}$ ($P_k \neq P_j$) may bar another set of t honest parties in $\mathcal{P}$, say the first t honest parties, to receive their respective points on $f_k(x)$ and $g_k(x)$. In the worst case, there can be t corrupted parties in $\mathcal{C}$, who may follow similar strategy as explained above and can ensure that *every honest* party in $\mathcal{P}$ waits indefinitely to receive their respective points on polynomials, corresponding to some corrupted party (ies) in $\mathcal{C}$. Thus *every honest* P_i in $\mathcal{P}$ may wait indefinitely to compute a_i and b_i.

The Technical Problem and Possible Solution: From the description of ACS (see section 12.1), it follows that ACS can be used to agree on a set of parties who will eventually satisfy property Q, where Q should have the following characteristic: *if some honest* P_i *concluded that some party* P_j *satisfies* Q*, then every other honest party will also eventually conclude that* P_j *satisfies* Q. However, in the steps given in Fig. 12.2, the parties use ACS to agree on a set of parties satisfying some property P which does not satisfy the above characteristic of Q. Specifically, in this case the property P is as follows: a party has selected two degree-t polynomials, sent one point on them to every party and A-casted 1. Now as explained above, a corrupted P_j may not give points to *all* honest parties and can still A-cast 1. So even if some *honest* party may receive points on the polynomials from P_j and concludes that P_j satisfies P, it does not mean that every other honest party will also conclude the same, as they may never receive values from P_j. It is this subtle property P in ACS, which causes the BSS scheme of [152] and hence the AMPC of [107] to fail to satisfy the **termination** (and **Correctness**) property.

A simple way to fix the above problem is to ask each $P_j \in \mathcal{P}$ to share two random values, say a^j and b^j using Sh protocol of some AVSS and then use ACS primitive to agree on a common set of $n-t$ parties $\mathcal{C}$ whose instances of Sh protocol will be eventually terminated by all (honest) parties in $\mathcal{P}$. Then each party P_i can locally compute $a_i = \sum_{P_j \in \mathcal{C}} a_i^j$ and $b_i = \sum_{P_j \in \mathcal{C}} b_i^j$, where a_i^j and b_i^j are i^{th} share of a^j and b^j respectively. Now by **termination** property of AVSS, every (honest) $P_i \in \mathcal{P}$ will eventually terminate Sh and thus will receive a_i^j, b_i^j corresponding to every $P_j \in \mathcal{C}$ and can compute a_i and b_i finally. However, the current best AVSS protocol with $n = 4t+1$ is due to [13], which requires a communication cost of $\mathcal{O}(\ell n^2 \log(|\mathbb{F}|)$ bits for concurrent sharing of ℓ secrets. If this AVSS is used then the resultant AMPC protocol will have a communication complexity of $\Omega(n^3 \log(|\mathbb{F}|)$ bits per multiplications gate. Hence to achieve a communication complexity of $\mathcal{O}(n^2 \log(|\mathbb{F}|)$ bits per multiplication gate, we require a different approach. We make an inroad towards this in next section by presenting a statistical protocol that generates $(t, 2t)$-$(1d)$-sharing.

12.3 Statistical Protocol for Generating $(t, 2t)$-$(1d)$-sharing of ℓ Secrets

We now present a novel protocol, called St-(t,2t)-(1d)-Share that allows a dealer $D \in \mathcal{P}$ (dealer can be any party from $\mathcal{P}$) to concurrently generate $(t, 2t)$-$(1d)$-sharing of $\ell \geq 1$ secrets from $\mathbb{F}$. We explain the idea of the protocol for a single secret s. D invokes St-AVSS-MS-Share to t-$(1d)$-share s. Let $f(x)$ be the degree-t polynomial used to t-$(1d)$-share s. D also invokes St-AVSS-MS-Share to $(2t-1)$-$(1d)$-share a random value r chosen from $\mathbb{F}$, which is independent of s. Let $g(x)$ be the degree-$(2t-1)$ polynomial used to $(2t-1)$-$(1d)$-share r. Now it is easy to see that $h(x) = f(x) + xg(x)$ will be a degree-$2t$ polynomial, such that $h(0) = s$. So if every party P_i locally computes $h(i) = f(i) + i \cdot g(i)$, then this will generate the $2t$-$(1d)$-sharing of s. Protocol St-(t,2t)-(1d)-Share follows this principle for all the ℓ secrets concurrently. The protocol is given in Fig. 12.3

Figure 12.3: Protocol for Generating $(t, 2t)$-$(1d)$-sharing of ℓ secrets Concurrently.

Protocol St-(t,2t)-(1d)-Share($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), \epsilon$)

Code for D: Only D executes this code

1. Invoke St-AVSS-MS-Share($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), t, \epsilon$) and St-AVSS-MS-Share($D, \mathcal{P}, R = (r^1, \ldots, r^\ell), 2t - 1, \epsilon$), where the elements of R are randomly chosen from $\mathbb{F}$.

Code for P_i: Every party executes this code

1. Participate in St-AVSS-MS-Share($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), t, \epsilon$) and St-AVSS-MS-Share($D, \mathcal{P}, R = (r^1, \ldots, r^\ell), 2t - 1, \epsilon$).
2. Wait to terminate St-AVSS-MS-Share($D, \mathcal{P}, S = (s^1, \ldots, s^\ell), t, \epsilon$) with i^{th} shares of $S = (s^1, \ldots, s^\ell)$, say $(\varphi_i^1, \ldots, \varphi_i^\ell)$. Wait to terminate St-AVSS-MS-Share($D, \mathcal{P}, R = (r^1, \ldots, r^\ell), 2t - 1, \epsilon$) with i^{th} shares of $R = (r^1, \ldots, r^\ell)$, say $(\chi_i^1, \ldots, \chi_i^\ell)$.
3. For $l = 1, \ldots, \ell$, locally compute $\psi_i^l = \varphi_i^l + i \cdot \chi_i^l$, output φ_i^l and ψ_i^l as i^{th} share of s corresponding to t-$(1d)$-sharing and $2t$-$(1d)$-sharing respectively and terminate St-(t,2t)-(1d)-Share.

We now prove the properties of protocol St-(t,2t)-(1d)-Share.

Theorem 12.3 *Protocol St-(t,2t)-(1d)-Share achieves the following properties:*

1. **Termination:** *(a) If D is honest, then all honest parties will eventually terminate St-(t,2t)-(1d)-Share. (b) If D is corrupted and some honest party terminates St-(t,2t)-(1d)-Share, then all honest parties will terminate the protocol, except with probability ϵ.*

2. **Correctness:** *(a) If D is honest, then all the ℓ secrets are correctly $(t, 2t)$-$(1d)$-shared among the parties in $\mathcal{P}$. (b) If D is corrupted and the honest parties terminate St-(t,2t)-(1d)-Share, then there are ℓ values, that are correctly $(t, 2t)$-$(1d)$-shared among the parties in $\mathcal{P}$, except with probability ϵ.*

3. **Secrecy:** *$\mathcal{A}_t$ will have no information about the secrets of an honest D.*

PROOF: **Termination:** First part of termination follows from **Termination 1** property of protocol St-AVSS-MS, following which, both the instances of St-AVSS-MS-Share will be eventually terminated by every honest party when D is honest. Hence every honest party will eventually terminate St-(t,2t)-(1d)-Share after performing the local computations. Similarly, second part of termination follows from **Termination 2** of St-AVSS-MS.

Correctness: First part of correctness is asserted as follows: When D is honest, then both the instances of St-AVSS-MS-Share will correctly generate t-$(1d)$-sharing and $(2t-1)$-$(1d)$-sharing of secrets without any error. Now the rest follows easily from the protocol steps. Now second part of correctness is proved as follows: When D is corrupted and the honest parties terminate St-(t,2t)-(1d)-Share, then both the instances of St-AVSS-MS-Share has correctly generated t-$(1d)$-sharing and $(2t-1)$-$(1d)$-sharing of secrets, *each* except with error probability at most ϵ. Hence $(t, 2t)$-$(1d)$-sharing of ℓ values will be correctly generated, except with probability at most ϵ.

Secrecy: From **Secrecy** property of St-AVSS-MS, the secret S will remain secure after the execution of St-AVSS-MS-Share that generates the t-$(1d)$-sharing of S. Now the way $2t$-$(1d)$-sharing of S is computed maintains the secrecy of S. Hence S remains secure in St-(t,2t)-(1d)-Share when D is honest. □

Theorem 12.4 *Protocol St-(t,2t)-(1d)-Share privately communicates $\mathcal{O}((\ell n^2 + n^3)\log|\mathbb{F}|)$ bits and A-casts $\mathcal{O}(n^3 \log \frac{1}{\epsilon})$ bits.*

Proof: Follows from Theorem 11.22 and the fact that St-(t,2t)-(1d)-Share invokes two instances of St-AVSS-MS-Share. □

12.4 Statistical AMPC Protocol with $n = 4t + 1$

Once we have an efficient protocol for generating $(t, 2t)$-$(1d)$-sharing, our AMPC protocol proceeds in the same way as that of [13]. Specifically, our AMPC protocol is a sequence of three phases: preparation, input and computation. In the preparation phase, corresponding to each multiplication and random gate, a $(t, 2t)$-$(1d)$-sharing of random secret will be generated. So in total $(t, 2t)$-$(1d)$-sharing of $c_M + c_R$ random values will be generated. In the input phase the parties t-$(1d)$-share their inputs and agree on a common set of at least $n - t$ parties who correctly t-$(1d)$-shared their inputs (every honest party will eventually get shares of the inputs of the parties in the common set). In the computation phase, based on the inputs of the parties in this common set, the actual circuit will be computed gate by gate, such that the output of the intermediate gates are always kept as secret and are t-$(1d)$-shared among the parties. Due to the linearity of the used t-$(1d)$-sharing, the linear gates can be computed locally without communication. Each multiplication gate will be evaluated with the help of the $(t, 2t)$-$(1d)$-sharing associated with it. For this, we adapt a technique from [52] used in synchronous settings, which is further used in AMPC of [13]. We now describe each of the three phases of our protocol.

12.4.1 Preparation Phase

The goal of the preparation phase is to generate $(t, 2t)$-$(1d)$-sharing of $c_M + c_R$ *secret* random values. For this, we design a protocol called St-Preparation, that asks each individual party to act as a dealer and $(t, 2t)$-share $\frac{c_M + c_R}{n-2t}$ random values. Then an instance of ACS is executed to agree on a common set C of $n - t$ parties, who have correctly $(t, 2t)$-$(1d)$-shared $\frac{c_M + c_R}{n-2t}$ values. Out of these $n - t$ parties, at least $n - 2t$ are honest, who have indeed $(t, 2t)$-$(1d)$-shared random values, which are unknown to $\mathcal{A}_t$. So if we consider the $(t,2t)$-$(1d)$-sharing done by the honest parties (each of them has done $\frac{c_M+c_R}{n-2t}$ $(t,2t)$-$(1d)$-sharing) in common set C, then we will get $\frac{c_M+c_R}{n-2t} * (n-2t) = c_M + c_R$ random $(t,2t)$-$(1d)$-sharing. For this, we use *Vandermonde Matrix* [52] and its ability to extract randomness which has been exploited in [141, 52, 13]. A brief discussion on *Vandermonde Matrix* was presented in Section 9.4.2 of Chapter 9.

We now present protocol St-Preparation in Fig. 12.4

Figure 12.4: Preparation Phase: Generation of $(t,2t)$-$(1d)$-sharing of $c_M + c_R$ secret random values.

Protocol St-Preparation($\mathcal{P}, \epsilon$)

Secret Sharing: Code for P_i: Every party executes this code

1. Select $L = \frac{c_M+c_R}{n-2t}$ random secret elements $(s^{(i,1)}, \ldots, s^{(i,L)})$ from $\mathbb{F}$. As a dealer, invoke St-(t,2t)-(1d)-Share($P_i, \mathcal{P}, S^i, \frac{\epsilon}{n}$) to generate $(t,2t)$-$(1d)$-sharing of $S^i = (s^{(i,1)}, \ldots, s^{(i,L)})$.
2. For $j = 1, \ldots, n$, participate in St-(t,2t)-(1d)-Share($P_j, \mathcal{P}, S^j, \frac{\epsilon}{n}$).

Agreement on a Common Set: Code for P_i: Every party executes this code

1. Create a set $C^i = \emptyset$. Upon terminating St-(t,2t)-(1d)-Share($P_j, \mathcal{P}, S^j, \frac{\epsilon}{n}$), include P_j in C^i.
2. Take part in ACS with the set C^i as input.

Generation of Random $(t,2t)$-$(1d)$-sharing: Code for P_i: Every party executes this code

1. Wait until ACS completes with output C containing $n-t$ parties. Obtain the i^{th} shares $\varphi_i^{(j,1)}, \ldots, \varphi_i^{(j,L)}$ corresponding to t-$(1d)$-sharing of S^j and i^{th} shares $\phi_i^{(j,1)}, \ldots, \phi_i^{(j,L)}$ corresponding to $2t$-$(1d)$-sharing of S^j for every $P_j \in C$. Without loss of generality, let $C = \{P_1, \ldots, P_{n-t}\}$.
2. Let V denote a $(n-t) \times (n-2t)$ publicly known *Vandermonde Matrix*.
   (a) For every $k \in \{1, \ldots, L\}$, let $(r^{(1,k)}, \ldots, r^{(n-2t,k)}) = (s^{(1,k)}, \ldots, s^{(n-t,k)})V$.
   (b) Locally compute i^{th} shares corresponding to t-$(1d)$-sharing of $r^{(1,k)}, \ldots, r^{(n-2t,k)}$ as $(\varsigma_i^{(1,k)}, \ldots, \varsigma_i^{(n-2t,k)}) = (\varphi_i^{(1,k)}, \ldots, \varphi_i^{(n-t,k)})V$.
   (c) Locally compute i^{th} shares corresponding to $2t$-$(1d)$-sharing of $r^{(1,k)}, \ldots, r^{(n-2t,k)}$ as $(\sigma_i^{(1,k)}, \ldots, \sigma_i^{(n-2t,k)}) = (\phi_i^{(1,k)}, \ldots, \phi_i^{(n-t,k)})V$ and terminate.

The values $r^{(1,1)}, \ldots, r^{(n-2t,1)}, \ldots, r^{(1,L)}, \ldots, r^{(n-2t,L)}$ denote the c_M+c_R random secrets which are $(t,2t)$-$(1d)$-shared.

Lemma 12.5 *Protocol St-Preparation satisfies the following properties:*

1. **Termination**: *All honest parties will eventually terminate St-Preparation, except with probability ϵ.*

2. **Correctness**: *The protocol correctly outputs $(t, 2t)$-$(1d)$-sharing of $c_M + c_R$ multiplication triples, except with probability ϵ.*

3. **Secrecy**: *The adversary $\mathcal{A}_t$ will have no information about $r^{(i,j)}$, for $i = 1, \ldots, n-2t$ and $j = 1, \ldots, \frac{c_M+c_R}{n-2t}$.*

PROOF: **Termination:** Here we first show that ACS will eventually output a set C containing $n-t$ parties (possibly containing some corrupted parties). According to the first part of **Termination** property of St-(t,2t)-(1d)-Share, all honest parties will eventually terminate the instance of St-(t,2t)-(1d)-Share initiated by every honest party. Now since there are at least $3t+1$ honest parties, even if the instances initiated by corrupted parties do not terminate at all, ACS will output a set C containing $n-t = 3t+1$ parties. But nevertheless, it may happen that C contains up to t corrupted parties as well.

Now we show that every honest party will eventually terminate the instance of St-(t,2t)-(1d)-Share initiated by the parties in C and therefore will receive the shares of the secrets shared by the parties in C eventually, except with error probability at most ϵ. For all the honest parties in C, the above will hold without any error probability. But there may be at most t corrupted parties in C. The instance of St-(t,2t)-(1d)-Share initiated by each of the corrupted parties in C will be terminated by every honest party, except with probability $\frac{\epsilon}{n}$. This follows from the second part of the **Termination** property of St-(t,2t)-(1d)-Share, according to which if some honest party terminates St-(t,2t)-(1d)-Share, then all honest parties will terminate the protocol, except with probability $\frac{\epsilon}{n}$ when the instance of St-(t,2t)-(1d)-Share is initiated by a corrupted party (remember that in St-preparation, the instances of St-(t,2t)-(1d)-Share has been executed with error parameter $\frac{\epsilon}{n}$). Now as there can be at most t corrupted parties in C, for all of them the instances initiated by them will be terminated by all honest parties eventually, except with error probability $t\frac{\epsilon}{n} \approx \epsilon$. Subsequently, the honest parties perform all the computations with the shares as specified in the protocol and finally terminate St-Preparation, except with error probability ϵ.

Correctness: According to the first part of **Correctness** of St-(t,2t)-(1d)-Share, the $(t, 2t)$-$(1d)$-sharing generated by every honest party in C will be correct without any error probability. For a corrupted party in C, the generated $(t, 2t)$-$(1d)$-sharing of secrets will be correct, except with error probability $\frac{\epsilon}{n}$. Since there can be at most t corrupted parties in C, for all of them, the generated $(t, 2t)$-$(1d)$-sharing will be correct, except with error probability $t\frac{\epsilon}{n} \approx \epsilon$. Hence protocol St-Preparation will correctly output $(t, 2t)$-$(1d)$-sharing of $c_M + c_R$ multiplication triples, except with probability ϵ.

Secrecy: Secrecy of St-Preparation follows from **Secrecy** of St-(t,2t)-(1d)-Share and randomness extraction property of Vandermonde matrix [141, 52, 13]. □

Lemma 12.6 *Protocol St-Preparation privately communicating $\mathcal{O}(((c_M+c_R)n^2 + n^4)\log|\mathbb{F}|)$ bits, A-casts $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits and requires one invocation of ACS.*

PROOF: In protocol St-Preparation, each party executes an instance of St-(t,2t)-(1d)-Share, by acting as a dealer, to $(t, 2t)$-$(1d)$-share $L = \frac{c_M + c_R}{n-2t}$ secrets. Substituting $\ell = L$ in Theorem 12.4, the total private communication of the protocol is $\mathcal{O}\left((Ln^3 + n^4)\log(|\mathbb{F}|)\right)$ bits. Since $L = \frac{c_M + c_R}{n-2t}$ and $n - 2t = \Theta(n)$, the total private communication of the protocol will be $\mathcal{O}(((c_M + c_R)n^2 + n^4)\log|\mathbb{F}|)$ bits. Moreover, the protocol will A-cast $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits and requires one invocation of ACS. □

12.4.2 Input Phase

In protocol St-Input, each $P_i \in \mathcal{P}$ acts as a dealer to t-$(1d)$-share his input X_i containing c_i elements from $\mathbb{F}$. So total number of inputs $c_I = \sum_{i=1}^{n} c_i$. To achieve this, party P_i t-$(1d)$-share his input X_i by acting as a dealer and executing St-AVSS-MS-Share. The asynchrony of the network does not allow the parties to wait for more than $n - t = 3t + 1$ parties to complete their instance of St-AVSS-MS-Share. In order to agree on a common set of parties whose instance of St-AVSS-MS-Share have terminated and whose inputs will be taken into consideration for computation (of the circuit), one instance of ACS is invoked. At the end, everyone considers the t-$(1d)$-sharing of all the inputs shared by parties, only in the common set. Protocol St-Input is now presented in Fig. 12.5.

Figure 12.5: Input Phase: Generation of t-$(1d)$-sharing of the Inputs.

Protocol St-Input($\mathcal{P}, \epsilon$)

Secret Sharing: Code for P_i: Every party executes this code

1. Having input X_i, invoke St-AVSS-MS-Share($P_i, \mathcal{P}, X_i, t, \frac{\epsilon}{n}$), as a dealer, to generate t-$(1d)$-sharing of the values in X_i.
2. For every $j = 1, \ldots, n$, participate in St-AVSS-MS-Share($P_j, \mathcal{P}, X_j, t, \frac{\epsilon}{n}$).

Agreement on a Common Set: Code for P_i: Every party executes this code

1. Create a set $C^i = \emptyset$. Upon terminating St-AVSS-MS-Share($P_j, \mathcal{P}, X_j, t, \frac{\epsilon}{n}$), add P_j in C^i.
2. Participate in ACS with the set C^i as input.

Output Generation: Code for P_i:

1. Wait until ACS completes with output C containing $n - t$ parties. Output the shares corresponding to t-$(1d)$-sharing of the inputs of the parties in C and terminate St-Input.

Lemma 12.7 *Protocol St-Input satisfies the following properties:*

1. **Termination**: *All honest parties will eventually terminate the protocol, except with probability ϵ.*

2. **Correctness**: *The protocol correctly outputs t-$(1d)$-sharing of inputs of the parties in agreed common set C, except with probability ϵ.*

3. **Secrecy**: *The adversary $\mathcal{A}_t$ will have no information about the inputs of the honest parties in set C.*

PROOF: **Termination:** Here we first show that ACS will eventually output a set C containing $n-t$ parties (possibly containing some corrupted parties). According to the first part of **Termination** property of St-AVSS-MS-Share, all honest parties will eventually terminate the instance of St-AVSS-MS-Share initiated by every honest party. Now since there are at least $3t+1$ honest parties, even if the instances of St-AVSS-MS-Share initiated by corrupted parties do not terminate at all, ACS will output a set C containing $n-t = 3t+1$ parties. But nevertheless, it may happen that C contains up to t corrupted parties as well.

Now we show that every honest party will eventually terminate the instance of St-AVSS-MS-Share initiated by the parties in C and therefore will receive the shares of the secrets shared by the parties in C eventually, except with error probability at most ϵ. For all the honest parties in C, the above will hold without any error probability. But there may be at most t corrupted parties in C. The instance of St-AVSS-MS-Share initiated by each of the corrupted parties will be terminated by every honest party, except with probability $\frac{\epsilon}{n}$. This follows from the second part of the **Termination** property of St-AVSS-MS-Share, according to which if some honest party terminates St-AVSS-MS-Share, then all honest parties will terminate the protocol, except with probability $\frac{\epsilon}{n}$ when the instance of St-AVSS-MS-Share is initiated by a corrupted party (remember that in St-Input, the instances of St-AVSS-MS-Share are executed with error parameter $\frac{\epsilon}{n}$). Now as there can be at most t corrupted parties in C, for all of them the instances initiated by them will be terminated by all honest parties eventually, except with error probability $t\frac{\epsilon}{n} \approx \epsilon$. Subsequently, the honest parties will terminate St-Input, except with error probability ϵ.

Correctness: The t-$(1d)$-sharing generated by every honest party in C will be correct without any error probability. For a corrupted party in C, the generated t-$(1d)$-sharing of secrets will be correct, except with error probability $\frac{\epsilon}{n}$. Since there can be at most t corrupted parties in C, for all of them, the generated t-$(1d)$-sharing will be correct, except with error probability $t\frac{\epsilon}{n} \approx \epsilon$. Hence protocol St-Input will correctly output t-$(1d)$-sharing of the values of the parties in C, except with probability ϵ.

Secrecy: Secrecy follows from **Secrecy** of St-AVSS-MS-Share. □

Lemma 12.8 *Protocol St-Input privately communicates $\mathcal{O}((c_I n^2 + n^4)\log|\mathbb{F}|)$ bits, A-casts $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ bits and requires one invocation of ACS.*

PROOF: Follows from the following facts: St-Input invokes (a) n instances of St-AVSS-MS-Share with $\ell = c_i$ for $i = 1, \ldots, n$ (this requires private communication of $\mathcal{O}((c_I n^2 + n^4)\log|\mathbb{F}|)$ bits and A-cast of $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$) (b) one instance of ACS. □

12.4.3 Computation Phase

Once the input phase is over, in the computation phase, the circuit is evaluated gate by gate, where all inputs and intermediate values are t-$(1d)$-shared. As soon as a party holds his shares of the input values of a gate, he joins the computation of the gate. Due to the linearity of the used t-$(1d)$-sharing, linear gates can be computed locally simply by applying the linear function to the shares. With every random gate, one random $(t, 2t)$-$(1d)$-sharing (from the preparation phase) is associated, whose t-$(1d)$-sharing is directly used as outcome of the random gate. With every multiplication gate, one random $(t, 2t)$-$(1d)$-sharing (from the preparation phase) is associated, which is then used to compute t-$(1d)$-sharing of the product, following the technique of Damgard et al. [52] in synchronous settings. Given a $(t, 2t)$-$(1d)$-sharing of a secret random value r (i.e., $[r]_{(t,2t)}$), the technique of Damgard et al. [52] allows to evaluate a multiplication gate at the cost of one reconstruction. The technique of [52] is as follows: Let $z = xy$, where x, y are the inputs of the multiplication gate, such that x, y are t-$(1d)$-shared, i.e. $[x]_t, [y]_t$. Moreover, let $[r]_{(t,2t)}$ be the $(t, 2t)$-$(1d)$-sharing associated with the multiplication gate, where r is a secret random value. Now for computing $[z]_t$, the parties compute $[\Lambda]_{2t} = [x]_t.[y]_t + [r]_{2t}$. Then Λ is privately reconstructed by every $P_i \in \mathcal{P}$. Now every party defines $[\Lambda]_t$ as the default sharing of Λ, e.g., the constant degree-0 polynomial Λ and computes $[z]_t = [\Lambda]_t - [r]_t$. The secrecy of z follows from [52, 13]. The above approach is also used in the computation phase of the AMPC protocol of [13]. We now present the protocol for computation phase in Fig. 12.6.

Lemma 12.9 *Given that protocol* St-Preparation *and* St-Input *satisfy their properties specified in Lemma 12.5 and Lemma 12.7 respectively, Protocol* St-Computation *satisfies the following, except with probability at most ϵ:*

1. **Termination***: All honest parties will eventually terminate the protocol.*
2. **Correctness***: Given t-$(1d)$-sharing of $c_M + c_R$ secret random triples, the protocol computes the outputs of the circuit correctly and privately.*

Proof: **Termination:** Given that protocol St-Preparation and St-Input satisfy their **Termination** property specified in Lemma 12.5 and Lemma 12.7 respectively, termination of protocol St-Computation follows from the finiteness of the circuit representing function f and the termination property of OEC.

Correctness: Protocol St-Preparation outputs proper t-$(1d)$-sharing of $c_M + c_R$ secret random triples, except with probability ϵ. Also protocol St-Input outputs proper t-$(1d)$-sharing of the inputs of the parties in common set C, except with probability ϵ. Hence protocol St-Computation will correctly compute the circuit with probability at least $(1 - \epsilon)$. □

Lemma 12.10 *Protocol* St-Computation *privately communicates $O(n^2(c_M + c_O) \log |\mathbb{F}|)$ bits*

Proof: Follows from the fact that in protocol St-Computation, $2c_M + c_O$ instances of OEC are executed, corresponding to c_M multiplication gates and c_O output gates. □

Figure 12.6: Computation Phase: Evaluation the Circuit.

Protocol St-Computation($\mathcal{P}, \epsilon$)

For every gate in the circuit: Code for P_i
Wait until i^{th} share of each of the inputs of the gate is available. Now depending on the type of gate, proceed as follows:

1. **Input Gate:** $[s]_t = \mathsf{IGate}([s]_t)$: Simply output s_i, the i^{th} share of s.

2. **Linear Gate:** $[z]_t = \mathsf{LGate}([x]_t, [y]_t, \ldots)$: Compute and output $z_i = \mathsf{LGate}(x_i, y_i, \ldots)$, the i^{th} share of $z = \mathsf{LGate}(x, y, \ldots)$, where $x_i, y_i, \ldots$ denotes i^{th} share of $x, y, \ldots$.

3. **Multiplication Gate:** $[z]_t = \mathsf{MGate}([x]_t, [y]_t, [r]_{(t,2t)})$:

   (a) Let $[r]_{(t,2t)}$ be the random $(t, 2t)$-$(1d)$-sharing associated with the multiplication gate. Also let $(\varphi_1, \ldots, \varphi_n)$ and $(\phi_1, \ldots, \phi_n)$ denote the t-$(1d)$-sharing and $2t$-$(1d)$-sharing of r, respectively.

   (b) Compute $\Lambda_i = x_i.y_i - \phi_i$ the i^{th} share of Λ which is now $2t$-$(1d)$-shared.

   (c) For $j = 1, \ldots, n$, privately send Λ_i to party P_j. Apply OEC on received Λ_j's to privately reconstruct Λ.

   (d) Compute and output $z_i = \Lambda - \varphi_i$, the i^{th} share of z.

4. **Random Gate:** $[R]_t = \mathsf{RGate}([r]_{(t,2t)})$: Let $[r]_{(t,2t)}$ be the random $(t, 2t)$-$(1d)$-sharing associated with the random gate. Also let $(\varphi_1, \ldots, \varphi_n)$ denote the t-$(1d)$-sharing of r. Output $R_i = \varphi_i$ as the i^{th} share of $R(= r)$.

5. **Output Gate:** $x = \mathsf{OGate}([x]_t)$: If P_α is entitled to receive x then privately send x_i, the i^{th} share of x to party P_α. If P_i is entitled to receive x then apply OEC on received x_j's and output x.

12.4.4 Our Statistical AMPC Protocol

Now our new statistical AMPC protocol called St-AMPC for evaluating function f which is represented by a circuit containing c_I, c_L, c_M, c_R and c_O input, linear, multiplication, random and output gates, is: (1). Invoke St-Preparation($\mathcal{P}, \epsilon$) (2). Invoke St-Input($\mathcal{P}, \epsilon$) (3). Invoke St-Computation($\mathcal{P}, \epsilon$).

Theorem 12.11 *Let $n = 4t + 1$. Then protocol St-AMPC satisfies the following properties:*

1. **Termination**: *Except with probability ϵ, all honest parties will eventually terminate the protocol.*

2. **Correctness**: *Except with probability ϵ, the protocol correctly computes the outputs of the circuit.*

3. **Secrecy**: *The adversary $\mathcal{A}_t$ will get no extra information about the inputs of*

the honest parties other than what can be inferred by the inputs and outputs of the corrupted parties.

4. **Communication Complexity**: *The protocol privately communicates* $\mathcal{O}(((c_I + c_M + c_R + c_O)n^2 + n^4)\log|\mathbb{F}|)$ *bits,* A-casts $\mathcal{O}(n^4 \log \frac{1}{\epsilon})$ *bits and requires 2 invocations of* ACS.

Proof: The proof follows from the properties of protocol St-Preparation, St-Input and St-Computation. □

In the sequel, we will present a protocol for perfect AMPC. For that we will first present a perfect protocol for generating $(t, 2t)$-$(1d)$-sharing of ℓ secrets.

12.5 Perfect Protocol for Generating $(t, 2t)$-$(1d)$-sharing of ℓ Secrets

We now present a protocol called Pf-(t,2t)-(1d)-Share that allows a dealer $D \in \mathcal{P}$ to concurrently generate $(t, 2t)$-$(1d)$-sharing of $\ell \geq 1$ secrets. Pf-(t,2t)-(1d)-Share is exactly same as St-(t,2t)-(1d)-Share. The only difference is that Pf-(t,2t)-(1d)-Share uses Pf-AVSS-MS-Share as black box and therefore does not involve any error probability. So we just state the following theorem for Pf-(t,2t)-(1d)-Share:

Theorem 12.12 *Protocol Pf-(t,2t)-(1d)-Share achieves the following properties:*

1. **Termination:** *(a) If D is honest, then all honest parties will eventually terminate Pf-(t,2t)-(1d)-Share. (b) If D is corrupted and some honest party terminates Pf-(t,2t)-(1d)-Share, then all honest parties will eventually terminate Pf-(t,2t)-(1d)-Share.*

2. **Correctness:** *(a) If D is honest, then all the ℓ secrets are correctly $(t, 2t)$-$(1d)$-shared among the parties in $\mathcal{P}$. (b) If D is corrupted and the honest parties terminate Pf-(t,2t)-(1d)-Share, then there are ℓ values, that are correctly $(t, 2t)$-$(1d)$-shared among the parties in $\mathcal{P}$.*

3. **Secrecy:** *$\mathcal{A}_t$ will have no information about the secrets of an honest D.*

Theorem 12.13 *Protocol Pf-(t,2t)-(1d)-Share privately communicates $\mathcal{O}(\ell n^2 \log|\mathbb{F}|)$ bits and A-cast $\mathcal{O}(n^2 \log n)$ bits.*

Proof: Follows from theorem 11.36 and the fact that Pf-(t,2t)-(1d)-Share invokes two instances of Pf-AVSS-MS-Share. □

12.5.1 Comparison with Existing Protocol for generating $(t, 2t)$-$(1d)$-sharing

In [13] the authors presented a perfectly secure protocol, that privately communicates $\mathcal{O}(\ell n^3 \log|\mathbb{F}|)$ bits and A-casts $\mathcal{O}(n^2 \log|\mathbb{F}|)$ bits to generate $(t, 2t)$-$(1d)$-sharing of ℓ secrets concurrently. Informally, the authors generated $(t, 2t)$-$(1d)$-sharing of a single value in *asynchronous settings* from t-$(1d)$-sharing of $3t+1$ random values in asynchronous settings. This is done as follows: Let $[r^0]_t, \ldots, [r^{3t}]_t$ be t-$(1d)$-sharing of $3t + 1$ random values. Let $p(x)$ be the degree-t polynomial whose $t+1$ coefficients are $r^0, \ldots, r^t$. Let $q(x)$ be the degree-$2t$ polynomial whose $2t+1$ coefficients are $r^0, r^{t+1} \ldots, r^{3t}$. It is to be noted that both $p(x)$ and $q(x)$ have common constant term (which is r^0). Now the parties jointly perform some computation such that every party P_i receives $p(i)$ and $q(i)$ at the end. This ensures that r^0 is $(t, 2t)$-$(1d)$-shared among the parties. In [13] the authors have generated t-$(1d)$-sharing of $3t+1$ random values by using their AVSS scheme, incurring a total private communication of $\mathcal{O}(n^3 \log |\mathbb{F}|)$ bits and A-cast of $\mathcal{O}(n^2 \log(|\mathbb{F}|))$ bits. Thus the protocol of [13] requires a private communication of $\mathcal{O}(n^3 \log |\mathbb{F}|)$ bits and A-cast of $\mathcal{O}(n^2 \log(|\mathbb{F}|))$ bits to generate $(t, 2t)$-$(1d)$-sharing of a *single* value.

Thus protocol Pf-(t,2t)-(1d)-Share gains a factor of $\Omega(n)$ in communication complexity for generating $(t, 2t)$-sharing in comparison to the protocol of [13]. In fact, it is this gain of $\Omega(n)$, which helps our perfect AMPC protocol to gain $\Omega(n)$ in communication complexity, compared to the AMPC of [13]. In Section 12.3, a protocol (St-(t,2t)-(1d)-Share) with same communication complexity as Pf-(t,2t)-(1d)-Share was given for generating $(t, 2t)$-$(1d)$-sharing. However protocol St-(t,2t)-(1d)-Share has negligible error probability in **correctness** and **termination**, where as Pf-(t,2t)-(1d)-Share involves no error probability.

12.6 Our Perfect AMPC Protocol Overview

Once we have an efficient protocol for generating $(t, 2t)$-$(1d)$-sharing, our perfect AMPC protocol proceeds in the same way as our statistical AMPC and perfect AMPC of [13]. Specifically, our AMPC protocol is a sequence of three phases: preparation, input and computation. In the preparation phase, corresponding to each multiplication and random gate, a $(t, 2t)$-$(1d)$-sharing of random secret will be generated. In the input phase the parties t-$(1d)$-share their inputs and agree on a common set of at least $n-t$ parties who correctly t-$(1d)$-shared their inputs. In the computation phase, based on the inputs of the parties in this common set, the actual circuit will be computed gate by gate, such that the output of the intermediate gates are always kept as secret and are t-$(1d)$-shared among the parties. We now elaborate on each of the three phases.

12.6.1 Preparation Phase

Our protocol for Preparation phase, called as Pf-Preparation is same as protocol St-Preparation except that it invokes Pf-(t,2t)-(1d)-Share in the place of St-(t,2t)-(1d)-Share and therefore does not involve any error probability. So we just state the following lemmas:

Lemma 12.14 *Protocol Pf-Preparation satisfies the following properties:*

1. **Termination**: *All honest parties will eventually terminate Pf-Preparation.*
2. **Correctness**: *The protocol correctly outputs $(t, 2t)$-$(1d)$-sharing of $c_M + c_R$ multiplication triples.*
3. **Secrecy**: *The adversary $\mathcal{A}_t$ will have no information about $r^{(i,j)}$, for $i = 1, \ldots, n-2t$ and $j = 1, \ldots, \frac{c_M+c_R}{n-2t}$.*

Lemma 12.15 *Protocol Pf-Preparation privately communicates $\mathcal{O}((c_M+c_R)n^2 \log |\mathbb{F}|)$ bits, A-casts $\mathcal{O}(n^3 \log n)$ bits and requires one invocation of ACS.*

12.6.2 Input Phase

Our protocol for Input phase, called as Pf-Input is same as protocol St-Input except that it invokes Pf-AVSS-MS-Share in the place of St-AVSS-MS-Share and therefore does not involve any error probability. So we just state the following lemmas:

Lemma 12.16 *Protocol Pf-Input satisfies the following properties:*

1. **Termination**: *All honest parties will eventually terminate the protocol.*
2. **Correctness**: *The protocol correctly outputs t-$(1d)$-sharing of inputs of the parties in agreed common set C.*
3. **Secrecy**: *The adversary $\mathcal{A}_t$ will have no information about the inputs of the honest parties in set C.*

Lemma 12.17 *Protocol Pf-Input privately communicates $\mathcal{O}(c_I n^2 \log |\mathbb{F}|)$ bits, A-casts $\mathcal{O}(n^3 \log n)$ bits and requires one invocation of ACS.*

12.6.3 Computation Phase

Our protocol for Computation phase, called as Pf-Computation is same as protocol St-Computation except that it does not involve any error probability. This is because the protocols for the preparation phase and input phase does not involve any error probability. Instead of repeating the protocol, we just state the following lemmas:

Lemma 12.18 *Given that protocol Pf-Preparation and Pf-Input satisfy their properties specified in Lemma 12.14 and Lemma 12.16 respectively, protocol Pf-Computation satisfies the following:*

1. **Termination**: *All honest parties will eventually terminate the protocol.*
2. **Correctness**: *Given t-$(1d)$-sharing of $c_M + c_R$ secret random triples, the protocol computes the outputs of the circuit correctly and privately.*

Lemma 12.19 *Protocol Pf-Computation privately communicates $\mathcal{O}((c_M n^2 + c_O n) \log |\mathbb{F}|)$ bits*

12.6.4 Our Perfect AMPC Protocol

Now our new perfect AMPC protocol called Pf-AMPC for evaluating function f which is represented by a circuit containing c_I, c_L, c_M, c_R and c_O input, linear, multiplication, random and output gates, is: (1). Invoke Pf-Preparation($\mathcal{P}$) (2). Invoke Pf-Input($\mathcal{P}$) (3). Invoke Pf-Computation($\mathcal{P}$).

Theorem 12.20 *Let $n = 4t + 1$. Then protocol Pf-AMPC satisfies the following properties:*

1. **Termination**: *All honest parties will eventually terminate the protocol.*
2. **Correctness**: *The protocol correctly computes the outputs of the circuit.*

3. **Secrecy**: *The adversary $\mathcal{A}_t$ will get no extra information about the inputs of the honest parties other than what can be inferred by the inputs and outputs of the corrupted parties.*

4. **Communication Complexity**: *The protocol privately communicates $\mathcal{O}(((c_I + c_M + c_R)n^2 + c_O n)\log|\mathbb{F}|)$ bits,* A-casts *$\mathcal{O}(n^3 \log n)$ bits and requires 2 invocations of* ACS.

PROOF: The proof follows from the properties of protocol Pf-Preparation, Pf-Input and Pf-Computation. □

12.7 Conclusion and Open Problems

In summary, in this chapter we have focused on the communication complexity of AMPC protocols with $4t+1$ parties. We have shown the following:

1. The statistical AMPC protocol proposed in [107] (the authors have claimed the communication complexity of the protocol as $\mathcal{O}(n^2 \log|\mathbb{F}|)$ bits per *multiplication gate*) is not correct.

2. We then propose a new statistical AMPC that communicates $\mathcal{O}(n^2 \log|\mathbb{F}|)$ bits per *multiplication gate*.

3. Finally, we propose a perfect AMPC that communicates $\mathcal{O}(n^2 \log|\mathbb{F}|)$ bits per *multiplication gate*. Our perfect AMPC protocol is optimally resilient and improves the communication complexity of the best known perfectly secure optimally resilient AMPC protocol of [13] by a factor of $\Omega(n)$.

We conclude this chapter with the following open question:

Open Problem 20 *How to further reduce the communication complexity of AMPC protocol with $4t+1$?*

Chapter 13

Efficient Statistical ABA Protocol With Non-Optimal Resilience

In this chapter, we show another important application of the perfect AVSS protocol presented in Chapter 11 by designing an ABA protocol with $n = 4t + 1$. In the previous chapter, we have used the perfect AVSS presented in Chapter 11 for designing our perfect AMPC protocol that provides a communication complexity of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits per multiplication gate (which we have shown to be the best so far in the literature). Recall that, for designing AMPC in the previous chapter, we used our perfect AVSS as a tool to generate t-$(1d)$-sharing and $2t$-$(1d)$-sharing of secrets. In this chapter, we use another property of our AVSS emphasized in section 11.8 and present an efficient ABA protocol with $n = 4t+1$ whose communication complexity is significantly better than the communication complexity of the only known existing ABA protocol of [66, 67] with $n = 4t+1$. Specifically, our ABA achieves an amortized communication complexity of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits for attaining agreement on a single bit, where $\mathbb{F}$ with $|\mathbb{F}| \geq n$ denotes the finite field over which our protocol performs all the computations. On the other hand, the only known ABA with $4t + 1$ proposed in [66, 67] communicates $\Omega(n^4 \kappa \log |\mathbb{F}|)$ bits for single bit message, where κ is the error parameter. Like the ABA of [66, 67], our protocol has constant expected running time and also our protocol is *almost-surely terminating* and non-optimal in resilience.

Even though our protocol is non-optimal in resilience, it will find applications in many distributed fault-tolerant protocols like AMPC that requires $n = 4t + 1$ parties for its error-free computation. It is well known that ABA acts as an useful primitive in AMPC. ABA protocols with optimal resilience i.e $n = 3t + 1$ may be adapted for $n = 4t + 1$ and can be used in AMPC with $4t + 1$ parties. But comparing our ABA for $3t + 1$ (which provides the best known communication complexity for optimally resilient ABA protocols) presented in Chapter 9, with our ABA for $4t + 1$, we see that the later provides better communication complexity. Furthermore, while our ABA in Chapter 9 is $(1 - \epsilon)$-terminating, our ABA presented in this chapter is *almost-surely terminating*. Hence it is always advantageous to use ABA designed with $n = 4t+1$ parties as black box in AMPC protocols with $n = 4t + 1$ (and other fault tolerant protocols with $n = 4t + 1$).

13.1 Introduction

13.1.1 The Network and Adversary Model

This is same as described in section 8.1.1. Recall that the set of parties is denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$ and t out of the n parties can be under the influence of a *computationally unbounded Byzantine (active) adversary*, denoted as $\mathcal{A}_t$. We emphasize that we use $n = 4t + 1$ in this chapter.

13.1.2 Our Motivation and Contribution

The communication complexity of BA protocol is one of its important parameters. In the literature, a lot of attention has peen paid to improve the communication complexity of BA protocols in synchronous settings (see for example [26, 44, 57, 134, 75]). Unfortunately, not too much attention has been paid to design communication efficient ABA protocols. Therefore, we have studied the communication efficiency of ABA with optimal resilience in Chapter 9 and presented efficient solution for the same. In this chapter, we study the communication efficiency of ABA with non-optimal resilience i.e with $n = 4t + 1$.

In this chapter, we present $(0, 0)$-ABA protocol with $n = 4t + 1$ whose *amortized* communication complexity for agreeing on a single bit is $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits of private communication[1] and A-cast, where $\mathbb{F}$ is the working field and $|\mathbb{F}| \geq n$. Specifically, our ABA requires private communication and A-cast of $\mathcal{O}(n^3 \log |\mathbb{F}|)$ bits for reaching agreement on $2t+1 = \Theta(n)$ bits *concurrently.* Our ABA protocol requires *constant* expected running time.

We compare our ABA with the only known $(0, 0)$-ABA protocol of [66, 67] with $n = 4t + 1$ which also has constant expected running time. The ABA of [39] *privately* communicates as well as A-casts $\Omega(n^4 \kappa \log |\mathbb{F}|)$ bits, where κ is the error parameter in correctness of the AVSS used by them. So our ABA shows considerable gain (by a factor of $\Omega(n^2\kappa)$) in communication complexity over the ABA of [66], while keeping all other properties in place. Moreover, our ABA attains better communication complexity than the ABA protocols with optimal resilience presented in Chapter 9.

As mentioned before, our efficient ABA will find applications in many distributed fault-tolerant protocols like AMPC that require $n = 4t + 1$ parties for their error-free computation. The following reasons surely assert that choosing our ABA with non-optimal resilience over our ABA with optimal resilience (presented in Chapter 9) in an AMPC with $n = 4t + 1$ is more appropriate: (a) our ABA with $n = 4t + 1$ is much better in terms of communication complexity in comparison to our ABA with optimal resilience; (b) our ABA with non-optimal resilience is *almost-surely terminating*, whereas the ABA with optimal resilience is $(1 - \epsilon)$-terminating.

Our construction of ABA protocol employs the perfect AVSS scheme with $n = 4t + 1$ (called Pf-AVSS-MS) presented in Chapter 11. We use the property of protocol Pf-AVSS-MS, mentioned in section 11.8. Specifically, the property of protocol Pf-AVSS-MS is as follows: The sharing phase protocol Pf-AVSS-MS-Share (of Pf-AVSS-MS) can commit to at most $\ell(t + 1)$ secrets simultaneously with a communication complexity of $\mathcal{O}(\ell n^2 \log |\mathbb{F}|)$ bits and A-cast of $\mathcal{O}(n^2 \log n)$ bits. We will substitute Pf-AVSS-MS-Share in the place of WAVSS-MS-Share in

[1] Communication over private channels

multi-bit common coin protocol presented in section 9.4.2 of Chapter 9. We will show that our new common coin protocol obtained this way will satisfy all the properties of multi-bit common coin protocol. Also our protocol will have no error in termination due to the use of perfect AVSS protocol Pf-AVSS-MS, as opposed to our common coin protocol presented in section 9.4.2 which was only $(1-\epsilon)$-terminating. Apart from this our new common coin will achieve better communication complexity. These two properties will lead to our new *almost-surely terminating* $(0,0)$-ABA with better communication complexity.

13.2 Our ABA protocol with Non-optimal Resilience

Our ABA with $n = 4t+1$ follows the same approach of our ABA with optimal resilience presented in section 9.4 of Chapter 9. We only modify the multi-bit common coin protocol of section 9.4 by substituting our perfect AVSS called Pf-AVSS-MS in the place of WAVSS-MS. Rest of the protocols i.e Vote and ABA-MB presented in section 9.4 remain the same, except that both the protocols are now executed with $n = 4t+1$ parties instead of $n = 3t+1$ and in ABA-MB, we replace Common-Coin-MB by our new common coin protocol.

13.2.1 A New and Efficient Common Coin Protocol for Multiple Bits with $n = 4t+1$

We now present our multi-bit common coin protocol with $n = 4t+1$ for the sake of completeness. Recall that in our multi-bit common coin protocol (of section 9.4), every party $P_i \in \mathcal{P}$ has to share n values and later these n values may have to be reconstructed. In section 9.4, the sharing phase of our proposed statistical AVSS protocol, called WAVSS-MS, has been used by P_i to simultaneously share n values and later the reconstruction phase of protocol WAVSS-MS has been used to reconstruct these values.

Now recall from section 11.8 that the sharing phase protocol Pf-AVSS-MS-Share can commit to $\ell(t+1)$ secrets simultaneously with a private communication complexity of $\mathcal{O}(\ell n^2 \log|\mathbb{F}|)$ bits and A-cast of $\mathcal{O}(n^2 \log n)$ bits (section 11.8 also talks about how this can be achieved). Hence every P_i has to invoke Pf-AVSS-MS-Share with $\ell = 4$. This will allow Pf-AVSS-MS-Share to share $\ell(t+1) = 4(t+1) = 4t+1+3 = n+3$ secrets in which parties may ignore the last three secrets and consider first n secrets. This will require a private communication and A-cast of $\mathcal{O}(n^2 \log|\mathbb{F}|)$ bits (as $|\mathbb{F}| \geq n$). We can then use Pf-AVSS-MS-Rec (the reconstruction phase protocol) for reconstruction of those n values which will require private communication of $\mathcal{O}(n^2 \log|\mathbb{F}|)$ bits. In the sequel, we will invoke Pf-AVSS-MS-Share and Pf-AVSS-MS-Rec in the following way: Pf-AVSS-MS-Share$(P_i, \mathcal{P}, (x_{i1}, \ldots, x_{in}))$ to mean that P_i commits to n secrets, $(x_{i1}, \ldots, x_{in})$ using Pf-AVSS-MS-Share; similarly Pf-AVSS-MS-Rec$(P_i, \mathcal{P}, (x_{i1}, \ldots, x_{in}))$ is invoked to reconstruct the n values. Now our common coin protocol, called Common-Coin-MB-Non-Op is same as Common-Coin-MB of section 9.4 where all the instances of sharing phase of AVSS (i.e WAVSS-MS-Share) is now replaced by Pf-AVSS-MS-Share and reconstruction phase of AVSS (i.e WAVSS-MS-Rec-public) is now replaced by Pf-AVSS-MS-Rec. For the sake of completeness, we now present our protocol Common-Coin-MB-Non-Op in Fig. 13.1.

Now slight modifications of the proofs of protocol Common-Coin-MB (in section 9.4.2) will lead to the proofs of protocol Common-Coin-MB-Non-Op. Hence,

Figure 13.1: Multi-Bit Common Coin Protocol using Protocol Pf-AVSS-MS-Share and Pf-AVSS-MS-Rec as Black-Boxes

Protocol Common-Coin-MB-Non-OP($\mathcal{P}$)

Code for P_i: — All parties execute this code

1. For $j = 1, \ldots, n$, choose a random value x_{ij} and execute Pf-AVSS-MS-Share($P_i, \mathcal{P}, (x_{i1}, \ldots, x_{in})$).
2. Participate in Pf-AVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn})$) for every $j \in \{1, \ldots, n\}$. We denote Pf-AVSS-MS-Share($P_j, \mathcal{P}, (x_{j1}, \ldots, x_{jn})$) by Pf-AVSS-MS-Share$_j$.
3. Upon terminating Pf-AVSS-MS-Share$_j$, A-cast "P_i `terminated` P_j".
4. Create a dynamic set $\mathcal{T}_i$. Add party P_j to $\mathcal{T}_i$ if "P_k `terminated` P_j" is received from the A-cast of at least $n - t$ P_k's. Wait until $|\mathcal{T}_i| = n - t$. Then assign $T_i = \mathcal{T}_i$ and A-cast "`Attach` T_i `to` P_i". We say that the secrets $\{x_{ji} | P_j \in T_i\}$ are the secrets `attached` to party P_i.
5. Create a dynamic set $\mathcal{A}_i$. Add party P_j to $\mathcal{A}_i$ if

   (a) "`Attach` T_j `to` P_j" is received from the A-cast of P_j and

   (b) $T_j \subseteq \mathcal{T}_i$.

   Wait until $|\mathcal{A}_i| = n - t$. Then assign $A_i = \mathcal{A}_i$ and A-cast "P_i `Accepts` A_i".
6. Create a dynamic set $\mathcal{S}_i$. Add party P_j to $\mathcal{S}_i$ if

   (a) "P_j `Accepts` A_j" is received from the A-cast of P_j and

   (b) $A_j \subseteq \mathcal{A}_i$.

   Wait until $|\mathcal{S}_i| = n - t$. Then A-cast "`Reconstruct Enabled`". Let H_i be the current content of $\mathcal{A}_i$.

   Halt all the instances of Pf-AVSS-MS-Share$_j$ for all P_j who are are not yet included in current $\mathcal{T}_i$. Later resume all such instances of Pf-AVSS-MS-Share$_j$'s if P_j is included in $\mathcal{T}_i$.
7. Wait to receive "`Reconstruct Enabled`" from A-cast of at least $n - t$ parties. Participate in Pf-AVSS-MS-Rec($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) for every $P_k \in \mathcal{T}_i$. We denote Pf-AVSS-MS-Rec($P_k, \mathcal{P}, (x_{k1}, \ldots, x_{kn}), \epsilon'$) by Pf-AVSS-MS-Rec$_k$. Notice that as on when new parties are added to $\mathcal{T}_i$, P_i participates in corresponding Pf-AVSS-MS-Rec.
8. Let $u = \lceil 0.87n \rceil$. Every party $P_j \in \mathcal{A}_i$ is `associated` with $n - 2t$ values, say $V_{j1}, \ldots, V_{j(n-2t)}$ in the following way. Let x_{kj} for every $P_k \in T_j$ has been reconstructed. Let X_j be the $n - t$ length vector consisting of $\{x_{kj} \mid P_k \in T_j\}$. Then set $(v_{j1}, \ldots, v_{j(n-2t)}) = X_j \cdot V^{(n-t,n-2t)}$, where $V^{(n-t,n-2t)}$ is an $(n - t) \times (n - 2t)$ Vandermonde Matrix. Now $V_{jl} = v_{jl} \mod u$ for $l = 1, \ldots, n - 2t$.
9. Wait until $n - 2t$ values `associated` with all the parties in H_i are computed. Now for every $l = 1, \ldots, n-2t$ if there exits a party $P_j \in H_i$ such that $V_{jl} = 0$, then set 0 as the l^{th} binary output; otherwise set 1 as the l^{th} binary output. Finally output the $n - 2t$ length binary vector.

we only state the following theorems. Before that we stress that Common-Coin-MB-Non-Op will not be $(1-\epsilon)$-terminating as protocol Common-Coin-MB. This is because Common-Coin-MB-Non-Op uses Pf-AVSS-MS that has no error in termination property. So Common-Coin-MB-Non-Op will have no error in termination.

Theorem 13.1 *Protocol Common-Coin-MB-Non-Op is a t-resilient multi-bit common coin protocol with $n - 2t = 2t + 1$ bits output for $n = 4t + 1$ parties.*

Theorem 13.2 *Protocol Common-Coin-MB-Non-Op privately communicates $\mathcal{O}(n^3 \log |\mathbb{F}|)$ bits and A-casts $\mathcal{O}(n^3 \log |\mathbb{F}|)$ bits.*

PROOF: Follows from the fact that Common-Coin-MB-Non-Op requires at most n invocations of Pf-AVSS-MS-Share and Pf-AVSS-MS-Rec protocols. □

13.2.2 Final ABA Protocol for Achieving Agreement on $2t+1$ bits Concurrently with $n = 4t + 1$

Now our ABA is same as protocol ABA-MB presented in section 9.4.3 with protocol Common-Coin-MB is being replaced by Common-Coin-MB-Non-Op. Protocol Vote presented in Chapter 9 for $n = 3t + 1$ can be extended for $n = 4t + 1$. Now due to the substitution of protocol Common-Coin-MB-Non-Op that has no error in termination, our new ABA called ABA-MB-Non-Op is *almost-surely terminating.* Furthermore, since Common-Coin-MB-Non-Op is better in terms of communication complexity than Common-Coin-MB, protocol ABA-MB-Non-Op attains better communication complexity than ABA-MB. Hence we now present the following theorem:

Theorem 13.3 (ABA for $2t + 1$ Bits) *Let $n = 4t + 1$. Then protocol ABA-MB-Non-Op is a t-resilient, $(0, 0)$-ABA protocol for n parties. The protocol terminates in constant expected time. The protocol allows the parties to reach agreement on $2t+1$ bits simultaneously and involves private communication and A-cast of $\mathcal{O}(n^3 \log |\mathbb{F}|)$, where $|\mathbb{F}| \geq n$.*

Corollary 13.3.1 *Protocol ABA-MB-Non-Op requires an amortized communication complexity of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits (private communication plus A-cast) for reaching agreement on a single bit, where $|\mathbb{F}| \geq n$.*

13.3 Conclusion

In this chapter we have designed an efficient ABA protocol with non-optimal resilience. Our ABA provides significantly better communication complexity than the only known ABA with $4t + 1$ proposed in [66, 67]. Our ABA also shows an important application of our perfect AVSS designed in Chapter 11. We conclude this chapter with the following natural open question:

Open Problem 21 *Can we improve the communication complexity of ABA protocol with $n = 4t + 1$ parties over the one presented in this chapter?*

Chapter 14

Communication Optimal Multi-Valued A-cast and ABA with Optimal Resilience

Broadcast (BC) and Byzantine agreement (BA) are considered as the most fundamental primitives for fault-tolerant distributed computing and cryptographic protocols. An important variant of BC and BA are Asynchronous BC and ABA, respectively. Asynchronous Broadcast (known as A-cast) and ABA are used as a building block in many asynchronous distributed cryptographic tasks, such as AMPC, AVSS etc. The A-cast and ABA protocols are carried out among n parties, pairwise connected by private and secure channels, where t out of the n parties can be under the influence of a *Byzantine (active)* adversary, having *unbounded computing power*.

Though all existing protocols for A-cast and ABA are designed for a single bit message, in real life applications typically A-cast and ABA are invoked on *long message* rather than on single bit. Therefore, it is important to design efficient *multi-valued* A-cast and ABA protocols (i.e protocols with *long message*) which extract several advantages offered by directly dealing with long messages and are far better than multiple invocations to existing protocols for single bit [72, 75]. In this chapter, we design new and highly efficient multi-valued A-cast and ABA protocols for long messages, based on access to the existing A-cast and ABA protocols for short messages. In brief, we present the following results:

1. For an error parameter κ, we design a new, multi-valued A-cast protocol with $n = 3t+1$ that requires a private communication of $\mathcal{O}(\ell n)$ bits for an ℓ bit message, where ℓ is sufficiently large (the exact bound on ℓ is mentioned later in this chapter). Our A-cast protocol uses the existing A-cast protocol of [29] as a black box for smaller size message. The protocol of [29] is the only known protocol for A-cast and it requires a private communication of $\mathcal{O}(n^2)$ bits for a single bit message where $n = 3t+1$.

2. For an error parameter κ, we design a new, multi-valued ABA protocol with $n = 3t+1$, which requires a private communication of $\mathcal{O}(\ell n)$ bits to agree on an ℓ bit message, where ℓ is sufficiently large (the exact bound on ℓ is mentioned later in this chapter). Our protocol uses the best known communication efficient ABA protocol presented in Chapter 9 of this thesis as a black box, which requires a private communication of $\mathcal{O}(n^7\kappa)$ bits to agree on a $(t+1)$ bit message.

3. We also note that both our A-cast and ABA protocols are *communication optimal*, optimally resilient (i.e designed with $n = 3t + 1$) and are strictly better than existing protocols in terms of communication complexity for sufficiently large ℓ.

Our protocols are based on several new ideas. Fitzi et al. [75] are the first to design *communication optimal* multi-valued Byzantine Agreement (BA) protocols for large message with the help of BA protocols for smaller message, in *synchronous network*. Achieving the same in asynchronous network was left as an interesting open question in [75]. Our results in this chapter marks a significant progress on the open problem by giving protocols with a communication complexity of $\mathcal{O}(\ell n)$ bits for large ℓ. Moreover, to the best of our knowledge, ours is the first ever attempt to design multi-valued A-cast and ABA protocols, using existing A-cast and ABA protocols (for small messages) as a black-box.

14.1 Introduction

The problem of Broadcast (BC) and Byzantine Agreement (BA) (also popularly known as consensus) were introduced in [132] and since then they have emerged as the most fundamental problems in distributed computing. They have been used as building blocks for several important secure distributed computing tasks such as MPC [3, 19, 5, 6, 7, 20, 12, 13, 14, 21, 9, 36, 41, 48, 49, 52, 95, 93, 98, 101, 103, 104, 135, 138, 143, 126], VSS [43, 55, 108, 9, 95, 20, 41, 62, 63, 137, 48, 21, 39, 138, 73, 91, 93, 109, 125, 12, 14, 98, 126, 50, 47, 35, 96, 28, 133, 66, 64, 8, 37, 22, 53, 92, 123, 145, 34, 97] etc. In practice, BC and BA are used in almost any task that involves multiple parties, like voting, bidding, secure function evaluation, threshold key generation etc.

Both BC and BA protocols are executed among a set $\mathcal{P}$ of n parties, who are connected to each other by pairwise secure channel. In brief, a BC protocol allows a special party in $\mathcal{P}$, called sender, to send some message identically to all other parties in $\mathcal{P}$, such that even when the sender is corrupted, all honest parties in $\mathcal{P}$ output the same message. The challenge lies in achieving the above task despite the presence of t faulty parties in $\mathcal{P}$ including the sender, who may deviate from the protocol arbitrarily. BA problem is slightly different from BC. BA among a set $\mathcal{P}$ of n parties, each having an input value, allows them to reach agreement on a common value even if t out of the n parties are faulty and try to prevent agreement among the non-faulty parties. The faulty behavior may range from simple mistakes to total breakdown to skillful adversarial talent. Attaining agreement on a common value is difficult as one does not know whom to trust. It is known that BC and BA in information theoretic settings tolerating t Byzantine faulty parries is possible iff $n \geq 3t + 1$ [132, 72].

BC and BA are so closely related that they are mutually reducible, i.e, given a BC protocol, we can always design a BA protocol using BC as black box and vice versa [72, 75]. Given a BA protocol, a BC protocol can be constructed as follows: First, the sender sends the message m to every party in $\mathcal{P}$. Then, the parties use the BA protocol to reach agreement on m. On the other hand, given a BC protocol, a BA protocol can be constructed as follows: First, every party P_i in $\mathcal{P}$ broadcasts his message m_i to all the parties using an instance of BC protocol. Then every party outputs the message that he has received most often. Note that the later reduction from BC to BA requires n invocations to the BC protocol in order to realize a BA protocol.

The BC and BA problems have been investigated extensively in various models, characterized by the synchrony of the network, privacy of the channels, computational power of the faulty parties and many other parameters [68, 18, 29, 39, 35, 118, 72, 110, 2, 24, 25, 26, 30, 31, 32, 44, 56, 54, 57, 59, 60, 61, 74, 70, 71, 65, 67, 78, 86, 89, 114, 117, 134, 136, 150, 148, 149]. But most of the emphasis was on the study of BC and BA in synchronous settings [68, 72, 110, 2, 24, 25, 31, 26, 44, 56, 54, 57, 59, 60, 61, 74, 70, 71, 65, 67, 78, 86, 89, 114, 117, 150, 148, 149], where almost all aspects of these problems has been studied. While the works of [69, 115, 87, 58, 122, 26, 25] focus on deriving round complexity lower bounds and designing round optimal protocols for BC and BA problems, the works of [57, 24, 23, 44, 75] deal with deriving communication complexity lower bounds and designing communication optimal protocols. But the limitations with the BC and BA protocols in synchronous settings are that they assume that the delay in the transmission of every message in the network is bounded by a fixed constant. Though these protocols are theoretically impressive, the above assumption is a very strong assumption in practice. This is because a delay in the transmission of even a single message may hamper the overall property of the protocol. Therefore, BC in asynchronous network, known as A-cast and BA in asynchronous network, known as ABA have been introduced and studied in the literature [18, 136, 29, 32, 30, 66, 39, 35, 1, 134]. In this chapter, we study A-cast and ABA problem, specifically the communication complexity of these problems. In the sequel, we first present the model that we use for our work and the formal definitions of A-cast and ABA. Subsequently, we will present the literature survey on A-cast and ABA. Lastly, we will elaborate on our contribution in this chapter.

14.1.1 The Network and Adversary Model

This is same as described in section 8.1.1. Here we recall that the set of parties is denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$ and t out of the n parties can be under the influence of a *computationally unbounded Byzantine (active) adversary*, denoted as $\mathcal{A}_t$. We emphasize that we use $n = 3t + 1$ in this chapter.

14.1.2 Definitions

We now formally define A-cast (Though A-cast was defined in Chapter 7, we repeat it here for ease of reference), ABA and their variants.

Definition 14.1 (A-cast [35]) *: Let Π be a protocol executed among the set of parties $\mathcal{P}$ and initiated by a special party caller sender $S \in \mathcal{P}$, having input m (the message to be sent). Π is an A-cast protocol tolerating $\mathcal{A}_t$ if the following hold, for every behavior of $\mathcal{A}_t$ and every input m:*

1. **Termination:**
   (a) *If S is honest, then all honest parties in $\mathcal{P}$ will eventually terminate Π;*
   (b) *If any honest party terminates Π, then all honest parties will eventually terminate Π.*
2. **Correctness:**

(a) *If the honest parties terminate* Π*, then they do so with a common output* m^**;*

(b) *Furthermore, if the sender* S *is honest then* $m^* = m$.

We now define (ϵ, δ)-A-cast protocol, where both ϵ and δ are negligibly small values (Recall the discussion presented in the beginning of section 1.5 for the meaning of negligible) and are called as error probabilities of the A-cast protocol. Moreover, we have $n = \mathcal{O}(\log \frac{1}{\epsilon})$ and $n = \mathcal{O}(\log \frac{1}{\delta})$ (follows from the definition of negligible, i.e $\epsilon \leq \frac{1}{2^{\alpha n}}$ and $\delta \leq \frac{1}{2^{\alpha n}}$ as mentioned in section 1.5).

Definition 14.2 (**(ϵ, δ)-A-cast**) *: An A-cast protocol* Π *is called* (ϵ, δ)*-A-cast protocol if :*

1. Π *satisfies* **Termination** *described in Definition 14.1, except with an error probability of* ϵ *and*
2. *Conditioned on the event that every honest party terminates* Π*, protocol* Π *satisfies* **Correctness** *property described in Definition 14.1, except with error probability* δ*.*

Both A-cast and (ϵ, δ)-A-cast can be executed for long messages. A-cast and (ϵ, δ)-A-cast for long messages are referred as *multi-valued* A-cast and *multi-valued* (ϵ, δ)-A-cast respectively.

The definition of ABA and (ϵ, δ)-ABA are given in Section 9.1 of Chapter 9.

As in the case of A-cast, both ABA and (ϵ, δ)-ABA can be executed for long message and these type of ABA protocols will be referred as *multi-valued* ABA and *multi-valued* (ϵ, δ)-ABA, respectively. The important parameters of any A-cast and ABA protocol are: (a) *Resilience* (b) *Communication Complexity* (c) *Computational Complexity:* and (d) *Running Time* (A detailed description of these parameters is provided in Chapter 1).

14.1.3 The History of Asynchronous Broadcast or A-cast

The only known protocol for A-cast is due to Bracha [29] and the protocol is a $(0, 0)$-A-cast protocol. The A-cast protocol of [29] was used as a black box in the ABA protocol of [29]. The $(0, 0)$-A-cast protocol of [29] was designed with $n = 3t + 1$ (this is optimal since in synchronous network, BC tolerating $\mathcal{A}_t$ is possible iff $n \geq 3t + 1$) and requires a communication complexity of $\mathcal{O}(n^2)$ bits to A-cast a *single bit* message in constant running time. To the best of our knowledge, there is no (ϵ, δ)-A-cast protocol for non-zero ϵ and/or δ.

14.1.4 The History of Asynchronous Byzantine Agreement (ABA)

From [132, 118], any ABA protocol tolerating $\mathcal{A}_t$ is possible iff $n \geq 3t + 1$. Thus any ABA protocol designed with $n = 3t+1$ parties is called as *optimally resilient*. A detailed literature survey on ABA was presented in Section 9.1 of Chapter 9. In Table 14.1, we just summarize the best known existing ABA protocols including the ABA protocols presented in Chapter 9.

Table 14.1: Summary of Best Known Existing ABA Protocols

Ref.	Type	Resilience	Communication Complexity (CC) in bits	Expected Running Time (ERT)
[29]	$(0,0)$-ABA	$t < n/3$	$\mathcal{O}(2^n)$	$\mathcal{C} = \mathcal{O}(2^n)$
[66, 67]	$(0,0)$-ABA	$t < n/4$	$\mathcal{O}((nt + t^7)\log\lvert\mathbb{F}\rvert)$ [a]	$\mathcal{C} = \mathcal{O}(1)$
[39, 35]	$(\epsilon,0)$-ABA	$t < n/3$	Private[b]: $\mathcal{O}(\mathcal{C}n^{11}(\log\kappa)^4)$ [c] A-cast[d]: $\mathcal{O}(\mathcal{C}n^{11}(\log\kappa)^2\log n)$	$\mathcal{C} = \mathcal{O}(1)$
[1]	$(0,0)$-ABA	$t < n/3$	Private: $\mathcal{O}(\mathcal{C}n^6\log\lvert\mathbb{F}\rvert)$ A-cast: $\mathcal{O}(\mathcal{C}n^6\log\lvert\mathbb{F}\rvert)$	$\mathcal{C} = \mathcal{O}(n^2)$
Chapter 9	$(\epsilon,0)$-ABA	$t < n/3$	Private: $\mathcal{O}(\mathcal{C}n^6\log\kappa)$ A-cast: $\mathcal{O}(\mathcal{C}n^6\log\kappa)$	$\mathcal{C} = \mathcal{O}(1)$
Chapter 9	Multi-valued [e] $(\epsilon,0)$-ABA	$t < n/3$	Private: $\mathcal{O}(\mathcal{C}n^5\log\kappa)$ A-cast: $\mathcal{O}(\mathcal{C}n^5\log\kappa)$	$\mathcal{C} = \mathcal{O}(1)$

[a] Here $\mathbb{F}$ is the finite field over which the ABA protocol of [66, 67] works. It is enough to have $\lvert\mathbb{F}\rvert \geq n$ and therefore $\log\lvert\mathbb{F}\rvert$ can be replaced by $\log n$. In fact in the remaining table, $\mathbb{F}$ bears the same meaning.

[b] Communication over private channels between pair of parties in $\mathcal{P}$.

[c] In this table, κ is the error parameter of protocols.

[d] Total number of bits that needs to be A-casted.

[e] This protocol reaches agreement on $(t+1)$ bits concurrently. Therefore, the amortized communication complexity for reaching agreement on a single bit is only $\mathcal{O}(\mathcal{C}n^4\log\kappa)$ bits of private as well as A-cast communication.

14.1.5 Multi-valued A-cast and ABA: Motivation of Our work

A-cast and ABA are the most important primitives in asynchronous distributed computing. However, in real-life applications typically A-cast and ABA protocols are invoked on *long messages* (whose size can be in gigabytes) rather than on single bit. Even in AMPC [21, 35, 13], where typically lot of A-cast and ABA invocations are required, many of the invocations can be parallelized and optimized to a single invocation with a long message. Hence A-cast and ABA protocols with long message, called *multi-valued* A-cast and ABA, are very relevant to many real life situations. All existing protocols for A-cast [29] and ABA [136, 18, 29, 66, 67, 39, 35, 1, 127] are designed for single bit message. A naive approach to design multi-valued A-cast and ABA for $\ell > 1$ bit message is to parallelize ℓ invocations of existing A-cast and ABA protocols dealing with single bit. This approach requires a communication complexity that is ℓ times the communication complexity of the existing protocols for single bit and hence is inefficient.

In synchronous network, researchers have tried to design multi-valued broadcast and BA protocol by making use of existing broadcast and BA protocol for small message, as a black-box. Turpin and Coan [150] are the first to report a multi-valued BC protocol based on the access to a BC protocol for short message (a brief description of how the reduction works can be found in [75, 72]). Recently, following the same approach, Fitzi et al. [75] have designed *communication optimal* BC and BA protocols for large message. While all existing synchronous BA protocols required a communication cost of $\Omega(\ell n^2)$ bits, the BA protocols of [75] require a communication complexity of $\mathcal{O}(\ell n + poly(n, \kappa))$ bits to agree on an ℓ bit message. For a sufficiently large ℓ, the communication complexity expression reduces to $\mathcal{O}(n\ell)$, which is a clear improvement over $\Omega(\ell n^2)$. Moreover, in [75] the authors have shown that their BA protocols are *communication optimal* for large ℓ. A brief discussion on the approach used in [75] for designing BA protocol is presented in section 14.3 of this chapter. However, the BA protocols of [75] involve a negligible error probability of $2^{-\Omega(\kappa)}$ in **Correctness**.

Designing *communication optimal* multi-valued A-cast and ABA protocols for large message based on the application of existing A-cast and ABA protocols for smaller message was left as an interesting open question in [75]. In this chapter, we make significant progress on the open question posed in [75], by designing *communication optimal* multi-valued A-cast and ABA protocols for sufficiently large messages. To the best of our knowledge, ours is the first ever attempt to design multi-valued A-cast and ABA protocols.

14.1.6 Contribution of This Chapter

In this chapter, we present communication optimal, optimally resilient, multi-valued A-cast and ABA protocols for long message, using the existing A-cast and ABA protocols for short message as black-box. Our protocols maintain the resilience of underlying black box protocols. However, even though the underlying black box protocols involve error probability in at most one property, our multi-valued protocols introduce negligible error probability in both the properties namely, **Termination** and **Correctness**. In summary, our contributions are:

1. A *communication optimal*, *optimally resilient* (ϵ, δ)-A-cast protocol with $n = 3t+1$ that requires a communication complexity of $\mathcal{O}(\ell n + n^4 + n^3\kappa)$ bits for an ℓ bit message. Our A-cast protocol uses the existing A-cast protocol of [29] as a black box for smaller size message. The protocol of [29] is the only known protocol for A-cast and it requires a private communication of $\mathcal{O}(n^2)$ bits for a single bit message where $n = 3t+1$. For sufficiently large ℓ (i.e., $\ell = \omega(n^2(n+\kappa))$), the communication complexity of our protocol is $\mathcal{O}(\ell n)$ bits, which is strictly better than the only known A-cast protocol of [29] in terms of communication complexity.

2. A *communication optimal*, *optimally resilient* (ϵ, δ)-ABA protocol (i.e. with $n = 3t+1$) that attains a communication complexity of $\mathcal{O}(\ell n + n^{10}\kappa + n^7\kappa^2)$ bits to agree on an ℓ bit message. Our protocol uses the best known communication efficient ABA protocol presented in Chapter 9 of this thesis as a black box, which requires a private communication of $\mathcal{O}(n^7\kappa)$ bits to agree on a $(t+1)$ bit message. For any $\ell = \omega(n^9\kappa + n^6\kappa^2)$, the communication complexity of our protocol becomes $\mathcal{O}(\ell n)$ bits which is strictly better than all exiting ABA protocols.

   In our protocol, we use *player-elimination* framework introduced in [98] in the context of MPC. So far player-elimination was used only in MPC and AMPC and hence our result shows the first non-MPC application of the technique. Apart from this, we use a novel idea to expand a set of $t+1$ parties, with all the honest party(ies) in it holding a common message m, to a set of $2t+1$ parties with all honest parties in it holding m. Moreover, the expansion process requires a communication complexity of $\mathcal{O}(\ell n + poly(n, \kappa))$ bits, where $|m| = \ell$. We hope that this technique may be useful in designing protocol for many other form of consensus/Byzantine Agreement problems in asynchronous network that aim to achieve good communication complexity.

3. In [75], it is shown that any BC or BA protocol in synchronous networks with $t \in \Omega(n)$, requires a communication complexity of $\Omega(n\ell)$ bits for an ℓ bit message. Obviously, this lower bound holds for asynchronous networks as well. Now it is easy to see that our protocols for A-cast and ABA are *communication optimal* for any $\ell = \omega(n^2(n + \kappa))$ and $\ell = \omega(n^9\kappa + n^6\kappa^2)$, respectively.

   The bound on ℓ for which our ABA is communication optimal, is dependent on the communication complexity of our black box ABA protocol for small message. So invention of a better ABA protocol (for small messages) than the ABA of Chapter 9 in terms of communication complexity would naturally lead to better bound on ℓ (for which our ABA protocol will be communication optimal). But designing such efficient ABA protocol is beyond the scope of this chapter.

In Table 14.2 and 14.3, we summarize the properties of our protocols and corresponding black box protocols. For multi-valued A-cast protocol, we use the only known A-cast protocol of [29] as black box. On the other hand, for multi-valued ABA we use the ABA protocol presented in Chapter 9 as the black box protocol. From Table 14.1, we find that the communication complexity of the ABA protocol of Chapter 9 is $\mathcal{O}(\mathcal{C}n^5 \log \kappa)$ bits for both private as well as A-cast communication. Now simulating the A-cast by the A-cast protocol of [29], we find that the ABA protocol of Chapter 9 requires a private communication of $\mathcal{O}(n^7\kappa)$ bits, to agree on a $t + 1$ bit message.

In Table 14.2, we also specify the lower bound on the value of ℓ for which our protocols are optimal and are strictly better than existing protocols. Though we have taken the best known protocols from literature to use as black box, we could have used any existing protocol. In Table 14.2 and 14.3, ERT stands for Expected Running Time and CC stands for Communication Complexity in bits.

Table 14.2: Our Contribution

Primitive	This Chapter				Lower Bound on ℓ
	Type	Resilience	CC	ERT	
A-cast	(ϵ, δ)	$t < n/3$	$\mathcal{O}(\ell n + n^4 + n^3\kappa)$	$\mathcal{O}(1)$	$\omega(n^2(n + \kappa))$
ABA	(ϵ, δ)	$t < n/3$	$\mathcal{O}(\ell n + n^{10}\kappa + n^7\kappa^2)$	$\mathcal{O}(n^2)$	$\omega(n^9\kappa + n^6\kappa^2)$

Table 14.3: Corresponding Black box Protocols and their Properties.

Primitive	Exiting Best Known Protocols (used as black box)				
	Ref.	Type	Resilience	CC	ERT
A-cast	[29]	$(0, 0)$	$t < n/3$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
ABA	Chapter 9	$(\epsilon, 0)$	$t < n/3$	$\mathcal{O}(n^7\kappa)$	$\mathcal{O}(1)$

To bound the error probability of **Termination** by ϵ and **Correctness** by δ, our protocols for A-cast and ABA work over a finite Galois field $\mathbb{F}$ with $\mathbb{F} =$

$GF(2^\kappa)$, where κ has to be determined using the relations $\epsilon \geq \frac{n\ell 2^{-\kappa}}{\kappa}$ and $\delta \geq \frac{n^2\ell 2^{-\kappa}}{\kappa}$. We assume that $\ell = poly(\kappa, n)$ and $\epsilon \leq \frac{\delta}{n}$. Now any field element from field $\mathbb{F}$ can be represented by κ bits. In order to bound the error probability of our A-cast or ABA protocol by some specific values of ϵ and δ, we find out the *minimum* value of κ that satisfies $\epsilon \geq \frac{n\ell 2^{-\kappa}}{\kappa}$ and the *minimum* value of κ that satisfies $\delta \geq \frac{n^2\ell 2^{-\kappa}}{\kappa}$. Then we take the minimum of the two values of κ as the final value for κ. The final value for κ will consequently determine the field $\mathbb{F}$ over which the protocol should work.

14.1.7 The Road-map

This chapter has mainly two sections, one for our A-cast and another for our ABA protocol. Section 14.2 presents our (ϵ, δ)-A-cast protocol and Section 14.3 presents our (ϵ, δ)-ABA protocol. Inside each section we present the tools that are used to design respective protocol. Finally, we conclude this chapter with concluding remarks and open problems in section 14.4.

14.2 Communication Optimal (ϵ, δ)-A-cast Protocol

Since our A-cast protocol is conceptually simpler than our ABA protocol, we first present our (ϵ, δ)-A-cast protocol in this section and then we will describe our (ϵ, δ)-ABA protocol in the next section. In addition, we use certain techniques in our A-cast protocol which will be applicable in our ABA protocol as well. So presenting these techniques in the context of A-cast will help to understand the same in the context of ABA. Our new (ϵ, δ)-A-cast protocol designed with $n = 3t + 1$, called Optimal-A-cast, allows a party $S \in \mathcal{P}$ to identically send his message $m \in \{0, 1\}^\ell$, to every (honest) party in $\mathcal{P}$. Now before presenting our protocol, we briefly describe existing tools used in it.

14.2.1 Tools Used

14.2.1.1 Bracha's $(0, 0)$-A-cast Protocol

Bracha's $(0, 0)$-A-cast protocol is already recalled in Chapter 7. For the ease of easy reference, we state the communication complexity of the protocol, named as Bracha-A-cast.

Theorem 14.3 ([35]) *Protocol Bracha-A-cast privately communicates $\mathcal{O}(|M|n^2)$ bits to A-cast an $|M|$ bit message.*

For this chapter we use a different convention for using the protocol.

Notation 14.4 (Convention for Using Bracha's A-cast Protocol) *In the rest of the chapter, we use the following convention: By saying that 'P_i Bracha-A-casts M', we mean that P_i as a sender, initiates Bracha-A-cast$(P_i, \mathcal{P}, M)$. Then by saying that 'P_j receives M from the Bracha-A-cast of P_i', we mean that P_j terminates the execution of Bracha-A-cast$(P_i, \mathcal{P}, M)$, with M as the output.*

14.2.1.2 Hash Function [75, 40]

A *keyed* hash function $\mathcal{U}_\kappa$ maps arbitrary strings in $\{0,1\}^*$ to κ bit string with the help of a κ bit random key. So $\mathcal{U}_\kappa : \{0,1\}^* \rightarrow \{0,1\}^\kappa$. The function $\mathcal{U}_\kappa$ can be implemented as follows: Let m and r be the input to $\mathcal{U}_\kappa$, where m is a ℓ bit string that need to be hashed/mapped and r is the hash key selected from $\mathbb{F}$. Without loss of generality, we assume that $\ell = \text{poly}(\kappa)$. Then m is interpreted as a polynomial $f_m(x)$ over $\mathbb{F}$, where the degree of $f_m(x)$ is $\lceil \ell/\kappa \rceil - 1$. For this, m is divided into blocks of κ bits and each block of κ bits is interpreted as an element from $\mathbb{F}$. Then these field elements are considered as the coefficients of $f_m(x)$ over $\mathbb{F}$. Finally, $\mathcal{U}_\kappa(m,r) = f_m(r)$. It is easy to see that $f_m(r)$ belongs to $\mathbb{F}$.

Theorem 14.5 (Collision Theorem [75]) *Let m_1 and m_2 be two different ℓ bit messages. Then the probability that $\mathcal{U}_\kappa(m_1,r) = \mathcal{U}_\kappa(m_2,r)$ for a randomly chosen hash key r is $\frac{\ell 2^{-\kappa}}{\kappa}(\leq \frac{\epsilon}{n} \leq \frac{\delta}{n^2})$.*

PROOF: Assume that m_1 and m_2 are represented by polynomials $f_1(x)$ and $f_2(x)$ respectively, each having degree $\lceil \ell/\kappa \rceil - 1$. Now $\mathcal{U}_\kappa(m_1,r) = \mathcal{U}_\kappa(m_2,r)$ implies that $f_1(r) = f_2(r)$ holds for random r. We now compute the error probability with which the above event i.e $f_1(r) = f_2(r)$ may happen. First, we note that polynomials $f_1(x)$ and $f_2(x)$ may intersect each other (i.e they evaluate to the same value) in at most $\lceil \ell/\kappa \rceil - 1$ values which is same as the degree of the polynomials. Now if the randomly selected r happens to be one among these $\lceil \ell/\kappa \rceil - 1$ values, then the event $f_1(r) = f_2(r)$ and consequently $\mathcal{U}_\kappa(m_1,r) = \mathcal{U}_\kappa(m_2,r)$ will hold. But the event that r is one among these $\lceil \ell/\kappa \rceil - 1$ values can happen with error probability $(\lceil \ell/\kappa \rceil - 1)\frac{1}{|\mathbb{F}|} \approx \frac{\ell 2^{-\kappa}}{\kappa}(\leq \frac{\epsilon}{n} \leq \frac{\delta}{n^2})$. Hence the theorem. □

14.2.1.3 Finding (n,t)-star Structure in a Graph [35, 19]

The definition (n,t)-star along with an algorithm for finding it in an undirected graph, has been described in Section 11.5 of Chapter 11. The algorithm is named as Find-STAR(H) where H denotes the complementary graph of G in which we want to find the (n,t)-star. For ease of reference, we just re-state the following lemmas for algorithm Find-STAR(H).

Lemma 14.6 ([35]) *If Find-STAR outputs $(\mathcal{C},\mathcal{D})$ on input graph H, then $(\mathcal{C},\mathcal{D})$ is an (n,t)-star in H.*

Lemma 14.7 ([35]) *Let H be a graph with $\mathcal{P}$ as its vertex set, containing an independent set of size $n-t$. Then algorithm Find-STAR always outputs an (n,t)-star, say $(\mathcal{C},\mathcal{D})$, in H.*

Lemma 14.8 ([35]) *The computational complexity of Algorithm Find-STAR is polynomial.*

14.2.2 Protocol Optimal-A-cast

We now present our (ϵ,δ)-A-cast protocol called Optimal-A-cast. Protocol Optimal-A-cast consists of the following three phases:

1. **Distribution Phase:** Here S sends the message to all the parties in $\mathcal{P}$.

2. **Verification & Agreement on CORE Phase:** Here the parties jointly perform some computation in order to verify the consistency of the messages received from S. In case of successful verification, the honest parties agree on a set of at least $n - t = 2t + 1$ parties called $CORE$[1], such that *the honest parties in $CORE$ have received same message from S with very high probability.*

3. **Output Phase:** Here the (honest) parties in $CORE$ propagate the common message held by them (which they have received from S) to all other (honest) parties in $\mathcal{P} \setminus CORE$.

Informal Description of First Two Phases: Informally, in **Distribution Phase**, S sends his message m to every party in $\mathcal{P}$. In **Verification & Agreement on CORE Phase**, party P_i on receiving a message, say m_i from S, computes n hash values of m_i corresponding to n *distinct random hash keys*, say $r_{i1}, \ldots, r_{in}$ chosen from $\mathbb{F}$. To enable P_j to check whether P_j's received message m_j is same as P_i's received message m_i, party P_i privately sends r_{ij} and $\mathcal{V}_{ij} = \mathcal{U}_\kappa(m_i, r_{ij})$ to P_j. Party P_j, on receiving these values from P_i, checks whether $\mathcal{V}_{ij} = \mathcal{U}_\kappa(m_j, r_{ij})$ (for honest S and P_i, it should hold). P_j Bracha-A-casts a confirmation signal $\mathtt{OK}(P_j, P_i)$ if the above check passes. Now based on the confirmation signals, a graph with the parties in $\mathcal{P}$ as vertex set is formed and applying Find-STAR on the graph, an (n,t)-star $(\mathcal{C}, \mathcal{D})$ is obtained. The $(\mathcal{C}, \mathcal{D})$ is then agreed among all the parties and $\mathcal{D}$ component of it is taken as $CORE$.

Achieving the agreement (among the honest parties) on a common $(\mathcal{C}, \mathcal{D})$ is a bit tricky in asynchronous network. Even though the *confirmations* are Bracha-A-casted by parties, parties may end up with different versions of $(\mathcal{C}, \mathcal{D})$ while attempting to generate them locally, due to the asynchronous nature of the network. We solve this problem by asking S to first compute $(\mathcal{C}, \mathcal{D})$ after receiving enough *confirmations* and then Bracha-A-cast $(\mathcal{C}, \mathcal{D})$. After receiving $(\mathcal{C}, \mathcal{D})$ from the Bracha-A-cast of S, individual party checks if the received $(\mathcal{C}, \mathcal{D})$ is indeed a valid (n,t)-star and then sets $CORE = \mathcal{D}$. The protocols for **Distribution Phase** and **Verification & Agreement on CORE Phase** are presented in Fig. 14.1. Before outlining **Output Phase**, we prove Lemma 14.9-14.11.

Lemma 14.9 *For a pair of honest parties (P_i, P_j), if P_i Bracha-A-casts $\mathtt{OK}(P_i, P_j)$ and P_j Bracha-A-casts $\mathtt{OK}(P_j, P_i)$, then $m_i = m_j$, except with error probability of at most $\frac{\epsilon}{n}$ or $\frac{\delta}{n^2}$.*

PROOF: It is easy to see that the lemma is true without any error if S is honest. So we prove the lemma when S is corrupted. Since an honest P_i Bracha-A-casted $\mathtt{OK}(P_i, P_j)$ and an honest P_j Bracha-A-casted $\mathtt{OK}(P_j, P_i)$, it must be the case that the tests $\mathcal{V}_{ji} \stackrel{?}{=} \mathcal{U}_\kappa(m_i, r_{ji})$ and $\mathcal{V}_{ij} \stackrel{?}{=} \mathcal{U}_\kappa(m_j, r_{ij})$ have passed for P_i and P_j, respectively. By **Collision Theorem** (see Theorem 14.5), the above statement implies that $m_i = m_j$, except with probability at most $\frac{\epsilon}{n}$ or $\frac{\delta}{n^2}$, as the hash keys r_{ij} and r_{ji} are completely random and unknown to corrupted S. $\square$

[1]Here the property and definition of CORE is completely different from the property of CORE used in some of the previous chapters in the context of AVSS.

Figure 14.1: Protocols for First Two Phases of Optimal-A-cast: **Distribution Phase** and **Verification & Agreement on CORE Phase**

First two Phases of Protocol Optimal-A-cast$(S, \mathcal{P}, m, \epsilon, \delta)$

Protocol Distribution$(S, \mathcal{P}, m, \epsilon, \delta)$**:** Distribution Phase

CODE FOR S: Send m to every $P_i \in \mathcal{P}$ over the private channels.

Protocol Verification$(S, \mathcal{P}, m, \epsilon, \delta)$**:** Verification & Agreement on CORE Phase

i. CODE FOR P_i: This code will be executed by every party in $\mathcal{P}$, including S.

1. Wait to receive a message containing ℓ bits from S. Denote it by m_i.
2. Upon receiving m_i, choose n random, distinct hash keys $(r_{i1}, \ldots, r_{in})$ from $\mathbb{F}$. For $j = 1, \ldots, n$, compute $\mathcal{V}_{ij} = \mathcal{U}_\kappa(m_i, r_{ij})$ and send $(r_{ij}, \mathcal{V}_{ij})$ to party P_j.
3. Upon receiving $(r_{ji}, \mathcal{V}_{ji})$ from P_j, check whether $\mathcal{V}_{ji} \stackrel{?}{=} \mathcal{U}_\kappa(m_i, r_{ji})$. If yes, then Bracha-A-cast $\texttt{OK}(P_i, P_j)$.
4. Construct an undirected graph G_i with $\mathcal{P}$ as vertex set. Add an edge (P_j, P_k) in G_i upon receiving
   (a) $\texttt{OK}(P_k, P_j)$ from the Bracha-A-cast of P_k and
   (b) $\texttt{OK}(P_j, P_k)$ from the Bracha-A-cast of P_j.

ii. CODE FOR S: This code will be executed only by S.

1. For every new receipt of some $\texttt{OK}(*, *)$ from some Bracha-A-cast, update G_S. If a new edge is added to G_S, then execute Find-STAR$(\overline{G_S})$ on current G_S. If this is the first time when Find-STAR$(\overline{G_S})$ returns an (n, t)-star $(\mathcal{C}, \mathcal{D})$, then stop any further computation and Bracha-A-cast $(\mathcal{C}, \mathcal{D})$. Finally assign $CORE = \mathcal{D}$ and $\overline{CORE} = \mathcal{P} \setminus CORE$. If no (n, t)-star is found so far, then wait to receive more $\texttt{OK}(*, *)$'s and repeat the above computation until an (n, t)-star is obtained.

iii. CODE FOR P_i: This code will be executed by every party in $\mathcal{P}$, including S.

1. Wait to receive $(\mathcal{C}, \mathcal{D})$ from the Bracha-A-cast of S such that $|\mathcal{C}| \geq n - 2t$ and $|\mathcal{D}| \geq n - t$.
2. After receiving $(\mathcal{C}, \mathcal{D})$, wait to receive all $\texttt{OK}(*, *)$'s and keep updating G_i, until $(\mathcal{C}, \mathcal{D})$ becomes a valid (n, t)-star in G_i. After that, assign $CORE = \mathcal{D}$ and $\overline{CORE} = \mathcal{P} \setminus CORE$.
3. Assign $m^* = m_i$ if $P_i \in CORE$. Here m^* is the message which will be agreed upon by all the honest parties in $\mathcal{P}$ at the end of Optimal-A-cast.

Lemma 14.10 *If S is honest, then the (honest) parties will eventually agree on $CORE$ of size at least $2t + 1$. Moreover, if one honest party decides on $CORE$, then every honest party will eventually decide on same $CORE$ even for a corrupted S.*

PROOF: An honest S will send identical m to every party in $\mathcal{P}$. Hence for every pair of honest parties (P_i, P_j), P_i and P_j will eventually Bracha-A-cast $\texttt{OK}(P_i, P_j)$ and $\texttt{OK}(P_j, P_i)$, respectively. Hence the nodes corresponding to honest parties will eventually form a clique (independent set) of size at least $2t+1$ in G_i ($\overline{G_i}$) graphs of every honest P_i. However, it may be possible that some corrupted parties are also present in the clique. But what ever may be the case, by Lemma 14.7, S will eventually find an (n,t)-star $(\mathcal{C}, \mathcal{D})$ in G_S and will Bracha-A-cast the same. Now by the property of Bracha-A-cast, eventually every honest party P_i will receive $(\mathcal{C}, \mathcal{D})$ from S and also the $\texttt{OK}$ signals such that $(\mathcal{C}, \mathcal{D})$ will be a valid (n,t)-star in G_i. Hence every honest party will agree on $(\mathcal{C}, \mathcal{D})$ and therefore will agree on $CORE = \mathcal{D}$ as well.

For the second part of lemma, suppose some honest party P_i has decided on a $CORE$. This implies that P_i has received $(\mathcal{C}, \mathcal{D})$ from the Bracha-A-cast of S, such that $(\mathcal{C}, \mathcal{D})$ is a valid (n,t)-star in G_i. By the property of Bracha-A-cast, the same will eventually happen for every other honest P_j, who will eventually receive $(\mathcal{C}, \mathcal{D})$ from S and the corresponding $\texttt{OK}$ signals such that $(\mathcal{C}, \mathcal{D})$ becomes a valid (n,t)-star in graph G_j. □

Lemma 14.11 *All honest parties in $CORE$ (if it is constructed) will possess same message m^*, except with error probability at most ϵ or $\frac{\delta}{n}$. Moreover if S is honest then $m^* = m$.*

PROOF: It is trivial to show that if S is honest then every honest party in $CORE$ will possess $m^* = m$. So we consider the case when S is corrupted. Recall that $CORE = \mathcal{D}$ for an (n,t)-star $(\mathcal{C}, \mathcal{D})$. By property of (n,t)-star, $|\mathcal{C}| \geq n-2t$ which is at least $t+1$ in our case. So there is at least one honest party in $\mathcal{C}$, say P_i. Now the honest P_i has edges with every party in $\mathcal{D}$ which implies that P_i's message m_i is equal to the message m_j of every honest P_j in $\mathcal{D}$ which in turn implies that all the honest parties in $\mathcal{D}$ or $CORE$ possess same message.

We now estimate the error probability of the above event. From Lemma 14.9, P_i's message m_i is identical to the message m_j of an honest party P_j in $\mathcal{D}$, except with probability at most $\frac{\epsilon}{n}$ or $\frac{\delta}{n^2}$. Therefore, P_i's message m_i is identical to the messages of all the honest parties in $\mathcal{D}$, except with probability at most $|H|\frac{\epsilon}{n}$ or $|H|\frac{\delta}{n^2}$, where H is the set of honest parties in $\mathcal{D}$. Now since $|H| \geq n-2t = t+1$, we have $|H|\frac{\epsilon}{n} \approx \epsilon$ and $|H|\frac{\delta}{n^2} \approx \frac{\delta}{n}$. This proves that honest parties in $\mathcal{D} = CORE$ hold common m^*, except with probability at most ϵ or $\frac{\delta}{n}$. □

Informal Description of **Output Phase***:* Once the parties agree on $CORE$, with all honest parties in it holding some common m^*, we need to ensure that m^* propagates to all (honest) parties in $\overline{CORE} = \mathcal{P} \setminus CORE$, in order to reach agreement on m^*. This is achieved in **Output Phase** (presented in Fig. 14.2) with the help of the parties in $CORE$. A simple solution could be to ask each party in $CORE$ to send his m^* to all the parties in $\overline{CORE}$, who can wait to receive $t+1$ same m^* and then accept m^* as the message. This solution will work as there are at least $t+1$ honest parties in $CORE$. But clearly, this requires a communication complexity of $\mathcal{O}(\ell n^2)$ bits (which violates out promised bound for Optimal-A-cast). Hence, we adopt a technique proposed in [75] for designing a BA protocol in synchronous settings with $n = |\mathcal{P}| = 2t+1$ parties. Now the technique proposed in [75] requires a set of parties, say $\mathcal{H} \subset \mathcal{P}$ such that all the honest parties in $\mathcal{H}$ hold the same message and the majority of the parties in $\mathcal{H}$ are honest. Under this condition the technique allows the set of honest parties in $\mathcal{P} \setminus \mathcal{H}$ to obtain the common message of the honest parties in $\mathcal{H}$ with a communication cost of $\mathcal{O}(\ell n)$ bits. In our context $CORE$ has all the properties of $\mathcal{H}$. Hence we adopt the technique of [75] in our context in the following way: Every $P_i \in CORE$ sets $d = t+1$ and $c = \lceil \frac{\ell+1}{d} \rceil$ and transforms his message m^* into a polynomial $p(x)$ of degree $d-1$ over $GF(2^c)$. Now if somehow a party $P_j \in \overline{CORE}$ receives d values on $p(x)$, then he can interpolate $p(x)$ and receive m^*. For this, party $P_i \in CORE$ sends i^{th} value on $p(x)$, namely $p_i = p(i)$ to every $P_j \in \overline{CORE}$. As the corrupted parties in $CORE$ may send wrong p_i, party P_j should be able to detect correct values. For this, every party $P_i \in CORE$ also sends hash values of $(p_1, \ldots, p_n)$ for a random hash key to every $P_j \in \overline{CORE}$. Now P_j can detect 'clean' (or correct) values with the help of the hash values and eventually P_j will receive d 'clean' values (possibly from $d = t+1$ honest parties in $CORE$) using which he can compute m^*.

Figure 14.2: Protocol for Last Phase of Optimal-A-cast: **Output Phase**

Last phase of Protocol Optimal-A-cast$(S, \mathcal{P}, m, \epsilon, \delta)$

Protocol Output$(S, \mathcal{P}, m, \epsilon, \delta)$**:** Output Phase

i. CODE FOR P_i: Every party in $\mathcal{P}$ will execute this code.

1. If $P_i \in CORE$, do the following to help the parties in $\overline{CORE}$ to compute m^*:
   (a) Set $d = t+1$ and $c = \lceil \frac{\ell+1}{d} \rceil$.
   (b) Interpret m^* as a polynomial $p(x)$ of degree $d-1$ over $GF(2^c)$. For this, divide m^* into blocks of c bits and interpret each block as an element from $GF(2^c)$. These elements from $GF(2^c)$ are the coefficients of $p(x)$.
   (c) Send $p_i = p(i)$ to every $P_j \in \overline{CORE}$, where p_i is computed over $GF(2^c)$.
   (d) For every $P_j \in \overline{CORE}$, choose a random distinct hash key R_{ij} from $\mathbb{F}$ and send $(R_{ij}, \mathcal{X}_{ij1}, \ldots, \mathcal{X}_{ijn})$ to P_j, where for $k = 1, \ldots, n$, $\mathcal{X}_{ijk} = \mathcal{U}_\kappa(p_k, R_{ij})$. Here, to compute $\mathcal{X}_{ijk}$, interpret p_k as a c bit string.
   (e) Terminate this protocol with m^* as output.
2. If $P_i \in \overline{CORE}$, do the following to compute m^*:
   (a) Call p_k received from party $P_k \in CORE$ as 'clean' if there are at least $t+1$ P_j's in $CORE$, corresponding to which $\mathcal{X}_{jik} = \mathcal{U}_\kappa(p_k, R_{ji})$ holds, where $(R_{ji}, \mathcal{X}_{ji1}, \ldots, \mathcal{X}_{jin})$ is received from $P_j \in CORE$.
   (b) Wait to receive d 'clean' p_k's and upon receiving, interpolate $d-1$ degree polynomial $p(x)$ using those 'clean' values, interpret m^* from $p(x)$ and terminate this protocol with m^* as output.

Lemma 14.12 *If the parties agree on $CORE$, then every honest party in $\mathcal{P}$ will eventually terminate protocol* Output*, except with probability at most ϵ.*

PROOF: We show that the lemma holds without any error probability when S is honest and with error probability ϵ, when S is corrupted.

1. *S is honest:* Let E_{CORE} and $E_{\overline{CORE}}$ be the events that the honest parties in $CORE$ and $\overline{CORE}$ (respectively) terminate. We show that $Prob(E_{CORE}) = Prob(E_{\overline{CORE}}) = 1$. From the steps of the protocol Output, the parties in $CORE$ will always terminate after performing the steps as mentioned in step 1(a)-1(d) of the protocol. Hence $Prob(E_{CORE}) = 1$ holds.

   To show that $Prob(E_{\overline{CORE}}) = 1$, consider an honest party P_i in $\overline{CORE}$. Clearly, P_i will terminate if it receives $d = t+1$ 'clean' values eventually. To assert that P_i will indeed receive $d = t+1$ 'clean' values, we first show that the value p_k received from every honest P_k in $CORE$ will be considered as 'clean' by P_i. Consequently, since there are $t+1$ honest parties in $CORE$, P_i will eventually receive $t+1$ 'clean' values even though the corrupted parties in $CORE$ never send any value to P_i. As the honest parties in $CORE$ have common m^*, they will generate same $p(x)$ and therefore same $p_k = p(k)$. Hence, $\mathcal{X}_{jik} = \mathcal{U}_\kappa(p_k, R_{ji})$ will hold, with respect to $(R_{ji}, \mathcal{X}_{jik})$ of every honest P_j in $CORE$. As there at least $d = t+1$ honest parties in $CORE$, this proves that p_k received from honest $P_k \in CORE$ will be considered as 'clean' by P_i. This proves $Prob(E_{\overline{CORE}}) = 1$.

   Finally, we have

   $$\begin{aligned} Prob(\text{Every honest party in } \mathcal{P} \text{ terminate}) &= Prob(E_{CORE} \cap E_{\overline{CORE}}) \\ &= Prob(E_{CORE}) \cdot Prob(E_{\overline{CORE}}) \\ &= 1 \cdot 1 = 1 \end{aligned}$$

2. *S is Corrupted:* Here we show that $Prob(E_{CORE}) = 1$ and $Prob(E_{\overline{CORE}}) = (1-\epsilon)$. Consequently, we will have $Prob(\text{Every honest party in } \mathcal{P} \text{ terminates}) = 1 \cdot (1-\epsilon) = (1-\epsilon)$. As in the case of *honest* S, the parties in $CORE$ will always terminate even when S is corrupted. This asserts that $Prob(E_{CORE}) = 1$. But on the other hand, here $Prob(E_{\overline{CORE}}) = (1-\epsilon)$. The reason is: From the previous case (i.e. when S is honest), if all the honest parties in $CORE$ holds common m^*, then $Prob(E_{\overline{CORE}}) = 1$ holds; but the parties in $CORE$ holds common m^*, except with probability ϵ (from Lemma 14.11). So, we have $Prob(E_{\overline{CORE}}) = Prob(\text{honest parties in } CORE \text{ hold common } m^*) \cdot Prob(E_{\overline{CORE}} \mid \text{honest parties in } CORE \text{ hold common } m^*) = (1-\epsilon).1 = (1-\epsilon)$. Hence, every honest party in $\mathcal{P}$ will eventually terminate protocol Output, except with probability at most ϵ.

Hence the lemma. □

Lemma 14.13 *Conditioned on the event that all the honest parties terminate* Output*, every honest party in $\mathcal{P}$ will output m^* in protocol* Output*, except with probability at most δ. Moreover, if S is honest then $m^* = m$.*

PROOF: We consider the following cases, namely (a) when S is honest and (b) when S is corrupted.

- *S is honest:* Consider the following two events, E_{CORE} and $E_{\overline{CORE}}$, where E_{CORE} is the event that *all the honest parties in $CORE$ output same m^** and $E_{\overline{CORE}}$ is the event that *all the honest parties in $\overline{CORE}$ output same $m^* = m$.* We now assert that $Prob(E_{CORE}) = 1$ and $Prob(E_{\overline{CORE}}|E_{CORE}) = (1-\delta)$.

  1. $Prob(E_{CORE}) = 1$: This follows from Lemma 14.11. Moreover, here $m^* = m$, where m is the message of S.
  2. $Prob(E_{\overline{CORE}}|E_{CORE}) = (1-\delta)$: Here we show that every honest $P_i \in \overline{CORE}$ will output m^*, except with probability $\frac{\delta}{n}$. This will assert that all the honest parties in $\overline{CORE}$ will output same m^*, except with error probability $|\overline{H}|\frac{\delta}{n}$ where $\overline{H}$ is the set of honest parties in $\overline{CORE}$. As $|\overline{H}|$ can be at most t, we have $|\overline{H}|\frac{\delta}{n} \approx \delta$.
     So let $P_i \in \overline{CORE}$ be an honest party. Now the p_k value of each honest $P_k \in CORE$ will be eventually considered as 'clean' value by honest P_i. This is because there are at least $t+1$ honest parties in $CORE$, who hold same m^* and therefore same $p(x)$ (and hence $p(k)$) when S is honest. So $\mathcal{X}_{jik} = \mathcal{U}_\kappa(p_k, R_{ji})$ will hold, with respect to $(R_{ji}, \mathcal{X}_{jik})$ of every honest P_j in $CORE$. A corrupted $P_k \in CORE$ may send $\overline{p_k} \neq p_k$ to P_i, but $\overline{p_k}$ will not be considered as a 'clean' value with probability at least $(1-\frac{\delta}{n^2})$. This is because, in order to be considered as 'clean' value, $\overline{p_k}$ should satisfy $\mathcal{X}_{jik} = \mathcal{U}_\kappa(\overline{p_k}, R_{ji})$ with respect to $(R_{ji}, \mathcal{X}_{jik})$ of at least $t+1$ P_j's from $CORE$. The test will fail with respect to an honest party from $CORE$ with probability $\frac{c2^{-\kappa}}{\kappa} \approx \frac{\delta}{n^3}$ according to **Collision Theorem** (see Theorem 14.5; putting $\ell = c$, where $c = \lceil\frac{l+1}{d}\rceil = \lceil\frac{l+1}{t}\rceil$). Thus though the test may pass with respect to all corrupted parties in $CORE$ (at most t), the test will fail for every honest party from $CORE$ with probability $(1-\frac{\delta}{n^3})^{|H|}$, where H is the set of honest parties in $CORE$. Now since $|H|$ can be $\Theta(t)$, we have $(1-\frac{\delta}{n^3})^{|H|} \approx (1-|H|\frac{\delta}{n^3}) \approx (1-\frac{\delta}{n^2})$. Now the probability that none of the wrong $\overline{p_k} \neq p_k$ sent by corrupted P_ks in $CORE$ will be considered as 'clean' by honest P_i is $(1-\frac{\delta}{n^2})^{\Theta(t)} \approx (1-\Theta(t)\frac{\delta}{n^2}) \approx (1-\frac{\delta}{n})$ (as there can be at most $\Theta(t)$ corrupted parties in $CORE$). Hence, honest P_i will reconstruct $p(x)$ using d 'clean' values (which he is bound to get eventually), except with probability $\frac{\delta}{n}$.
     It is easy to see that $m^* = m$ here.

  Finally, we have

$$\begin{aligned} Prob(\text{Each honest party in } \mathcal{P} \text{ holds } m^*) &= Prob(E_{CORE} \cap E_{\overline{CORE}}) \\ &= Prob(E_{CORE}) \cdot Prob(E_{\overline{CORE}}|E_{CORE}) \\ &= 1 \cdot (1-\delta) = (1-\delta) \end{aligned}$$

- *S is corrupted:* Here also we consider same two events E_{CORE} and $E_{\overline{CORE}}$ and show that $Prob(E_{CORE}) = (1-\frac{\delta}{n})$ and $Prob(E_{\overline{CORE}}|E_{CORE}) = (1-\delta)$.

  1. $Prob(E_{CORE}) = (1-\frac{\delta}{n})$: Follows from Lemma 14.11.
  2. $Prob(E_{\overline{CORE}}|E_{CORE}) = (1-\delta)$: Follows from the case when S is honest.

So we have, $Prob($ Every honest party in $\mathcal{P}$ holds $m^*) = (1 - \frac{\delta}{n}) \cdot (1 - \delta) \approx (1 - \delta)$.

Hence the lemma holds irrespective of when S is honest or corrupted. $\square$

Now our new (ϵ, δ)-A-cast protocol called Optimal-A-cast is presented in Fig. 14.3.

Figure 14.3: Protocol Optimal-A-cast: Communication Optimal A-cast protocol.

Optimal-A-cast$(S, \mathcal{P}, m, \epsilon, \delta)$

1. Execute Distribution$(S, \mathcal{P}, m, \delta, \epsilon)$;
2. Execute Verification$(S, \mathcal{P}, m, \epsilon, \delta)$;
3. Execute Output$(S, \mathcal{P}, m, \epsilon, \delta)$.
4. Terminate Optimal-A-cast after terminating Output.

We now prove the properties of protocol Optimal-A-cast.

Theorem 14.14 *Protocol Optimal-A-cast is a (ϵ, δ)-A-cast protocol with $\epsilon \geq \frac{n\ell 2^{-\kappa}}{\kappa}$ and $\delta \geq \frac{n^2\ell 2^{-\kappa}}{\kappa}$.*

PROOF: **Termination: Termination (a)** is asserted as follows: If S is honest then by Lemma 14.10 all honest parties will agree on $CORE$ eventually and by Lemma 14.12, every honest party in $\mathcal{P}$ will terminate eventually. Now **Termination (b)** is proved as follows: If an honest party terminates protocol Optimal-A-cast, then it implies that it has terminated Output which in turn implies it has agreed on some $CORE$. Now by Lemma 14.10, if some honest party agrees on some $CORE$, then every other honest party will agree on the same $CORE$ eventually. Now by Lemma 14.12, if every honest party agrees on $CORE$, then all the honest party will terminate Output (and thus Optimal-A-cast), except with probability ϵ.

Correctness: Completely follows from Lemma 14.13. $\square$

Theorem 14.15 *Optimal-A-cast requires a private communication of $\mathcal{O}(\ell n + n^4 + n^3\kappa)$ bits.*

PROOF: Protocol Distribution requires ℓn bits of private communication. Protocol Verification requires $\mathcal{O}(n^2\kappa)$ bits of private communication and $\mathcal{O}(n^2)$ bits of Bracha-A-cast (which in turn requires $\mathcal{O}(n^4)$ bits of private communication). So Verification privately communicates $\mathcal{O}(n^2\kappa + n^4)$ bits. Protocol Output requires $\mathcal{O}(n^2 c + n^3\kappa)$ bits of private communication. Now $\mathcal{O}(n^2 c + n^3\kappa) = \mathcal{O}(n\ell + n^3\kappa)$ as $c = \lceil \frac{\ell+1}{d} \rceil = \lceil \frac{\ell+1}{t+1} \rceil$ and $n = \Theta(t)$. So protocol Optimal-A-cast requires a private communication of $\mathcal{O}(\ell n + n^4 + n^3\kappa)$ bits. $\square$

14.3 Communication Optimal (ϵ, δ)-**ABA** Protocol

We now present a novel, optimally resilient, communication optimal (ϵ, δ)-ABA protocol with $n = 3t + 1$, called Optimal-ABA. The protocol allows the honest parties in $\mathcal{P}$, each having input message of ℓ bits, to reach agreement on a common message $m^* \in \{0, 1\}^\ell$ containing ℓ bits. Moreover, if all the honest parties have same input m, then all the honest parties agree on m. We first describe the existing tools used in Optimal-ABA.

14.3.1 Tools Used

14.3.1.1 AVSS

So far the best known communication efficient statistical AVSS with $n = 3t + 1$ for dealing with single secret is SAVSS (consisting of sub-protocols (SAVSS-Share,SAVSS-Rec-Private)) presented in Chapter 8 of this thesis. We will use this protocol for our ABA. Assuming ρ to be the error probability of protocol SAVSS) and protocol SAVSS works over $\mathbb{F} = GF(2^\kappa)$, the communication complexity of protocol SAVSS is recalled as follows:

Theorem 14.16 (Communication Complexity of SAVSS)

- *Protocol SAVSS-Share incurs a private communication of $\mathcal{O}(n^4\kappa^2)$ bits and A-cast of $\mathcal{O}(n^3 \log n)$ bits.*
- *Protocol SAVSS-Rec-Private incurs a private communication of $\mathcal{O}(n^4\kappa^2)$ bits for P_α-private-reconstruction.*

After simulating the A-cast with the protocol of [29], we find that SAVSS-Share requires a private communication $\mathcal{O}(n^4(\kappa^2 + n \log n))$ bits to commit a secret from $\mathbb{F}$.

14.3.1.2 Agreement on a Common Subset (ACS)

Though ACS has been discussed in Chapter 12, we recast it as per the requirement of this chapter and present an elaborate discussion on it. In our (ϵ, δ)-ABA protocol, we come across the following situation: There exists a set of parties $\mathcal{R} \subseteq \mathcal{P}$ with $|\mathcal{R}| \geq t + 1$, such that each party in $\mathcal{R}$ is asked to A-cast (or SAVSS-Share) some value(s). The following discussion holds for both A-cast and SAVSS-Share. While the honest parties in $\mathcal{R}$ will eventually do the A-cast (SAVSS-Share), the corrupted parties in $\mathcal{R}$ may or may not do the same. So the (honest) parties in $\mathcal{P}$ want to agree on a *common* set $\mathcal{T} \subset \mathcal{R}$, with $1 \leq |\mathcal{T}| \leq |\mathcal{R}| - t$, such that A-cast (SAVSS-Share) instance of each party in $\mathcal{T}$ will be eventually terminated by the (honest) parties in $\mathcal{P}$. For this, the parties use ACS primitive (stands for Agreement on Common Subset), presented in [21]. For the sake of completeness, the ACS protocol, along with its properties is given in Fig. 14.4.

Theorem 14.17 ([21]) *Using protocol ACS, the (honest) parties in $\mathcal{P}$ can agree on a common subset $\mathcal{T}$ of $\mathcal{R}$ containing $1 \leq |\mathcal{T}| \leq |\mathcal{R}| - t$ parties, whose instances of A-cast (SAVSS-Share) will be eventually terminated by all the (honest) parties in $\mathcal{P}$.*

Figure 14.4: Protocol for Agreement on a Common Subset with $n = 3t + 1$

$\mathcal{T}$ = Protocol ACS($\mathcal{R}, |\mathcal{T}|$)

Assumption: For the ease of presentation, we assume that $\mathcal{R}$ contains the first $|\mathcal{R}|$ parties from $\mathcal{P}$.

Code for Party P_i: Each party in $\mathcal{P}$ executes this code

1. For each $P_j \in \mathcal{R}$ whose instance of A-cast (SAVSS-Share) is terminated by you, participate in ABA_j with input 1. Here for $j = 1, \ldots, |\mathcal{R}|$, ABA_j denotes the instance of ABA executed on behalf of $P_j \in \mathcal{R}$ to decide whether $P_j \in \mathcal{T}$.
2. Upon terminating $|\mathcal{T}|$ instances of ABA with output 1, enter input 0 to all other instances of ABA, for which you haven't entered a value yet.
3. Upon terminating all the $|\mathcal{R}|$ ABA protocols, let your $SubSet_i$ be the set of all indices j for which ABA_j had output 1.
4. Let $\mathcal{T}_i$ be the set of $|\mathcal{T}|$ parties corresponding to smallest $|\mathcal{T}|$ indices in $SubSet_i$. Output $\mathcal{T}_i$ and terminate ACS.

PROOF: We first show that at least $|\mathcal{T}|$ ABAs terminate with output 1. Suppose some honest party $P_i \in \mathcal{P}$ has entered a 0 value into some ABA_j in step 2 of the protocol. This means that at least $|\mathcal{T}|$ ABAs terminated with output 1. This is what we want to show. So assume that no honest party has entered a 0 value to any of the $|\mathcal{R}|$ ABAs. Since every honest $P_h \in \mathcal{R}$ will initiate his instance of A-cast (SAVSS-Share), by the **termination** property of A-cast (SAVSS-Share), every honest party in $\mathcal{P}$ will eventually terminate this instance of A-cast (SAVSS-Share). In other words, the input of all honest parties to ABA_h will be eventually 1. So by the **correctness** property of ABA, the output of ABA_h will be 1. As this is true with respect to every honest party in $\mathcal{R}$ and since there are at least $|\mathcal{R}| - t$ honest parties in $\mathcal{R}$, at least $|\mathcal{T}|$ ABAs will eventually terminate with output 1. This is because $|\mathcal{T}| \leq |\mathcal{R}| - t$.

Next we show that all $|\mathcal{R}|$ ABAs will eventually terminate. As $|\mathcal{T}|$ ABAs will eventually terminate with output 1, all the honest parties will eventually enter an input to the remaining ABAs and thus they will also eventually terminate.

It is clear from the protocol that once an honest party in $\mathcal{P}$ terminates Protocol ACS, he has a proper subset $\mathcal{T}$ of $\mathcal{R}$ and from the properties of ABA, it follows that all the honest parties in $\mathcal{P}$ will agree on the same $\mathcal{T}$.

Finally, it remains to be shown that every honest party in $\mathcal{P}$ will eventually terminate the instance of A-cast (SAVSS-Share), initiated by every $P_j \in \mathcal{T}$. This is true because $P_j \in \mathcal{T}$ implies that ABA_j must have terminated with output 1, which further implies that at least one honest party in $\mathcal{P}$, say P_i, must have entered 1 as his input in ABA_j. Now P_i must have entered 1 as his input in ABA_j because P_i has terminated the A-cast (SAVSS-Share) of P_j. Now by the **termination** property of A-cast (SAVSS-Share), every other honest party will also eventually terminate the A-cast (SAVSS-Share) of P_j. $\square$

Theorem 14.18 *The communication complexity of protocol* ACS *is same as that of* $|\mathcal{R}|$ *executions of ABA protocol.*

As shown in Fig. 14.4, protocol ACS protocol uses $|\mathcal{R}|$ instances of ABA protocol. If we use the best known communication efficient $(\rho, 0)$-ABA of Chapter 9 of this thesis (we use ρ instead of ϵ in order to avoid confusion), then we get a $(|\mathcal{R}|\rho, 0)$-ACS protocol. The reason is that the ABA protocol corresponding to each $P_i \in \mathcal{R}$ will terminate with probability $(1-\rho)$ and therefore all the ABA protocols corresponding to all the parties in $\mathcal{R}$ will terminate with probability $(1-\rho)^{|\mathcal{R}|} \approx (1-|\mathcal{R}|\rho)$ implying that the ACS protocol will terminate with probability $(1-|\mathcal{R}|\rho)$. Evidently, to obtain $(\rho, 0)$-ACS protocol, we can invoke the ABA of Chapter 9 with error parameter $\frac{\rho}{\mathcal{R}}$. Since there is no error in correctness of the ABA of Chapter 9, there will not be any error in correctness of our ACS protocol.

Notation 14.19 (Convention for Using ACS Protocol) *We will invoke our* $(\rho, 0)$-ACS *as* Patra-ACS$(\mathcal{R}, |\mathcal{T}|, \rho)$ *to agree on* $\mathcal{T}$. *We will set appropriate value for* ρ *as per the requirement in our communication optimal ABA protocol.*

Now we estimate the communication complexity of protocol Patra-ACS. According to Theorem 14.18, the communication complexity of Patra-ACS will be same as that of $|\mathcal{R}|$ executions of the ABA protocol of Chapter 9. Assuming that $\rho = 2^{-\Omega(\kappa)}$ and the multi-valued $(\rho, 0)$-ABA of Chapter 9 works over field $\mathbb{F}$, it requires $\mathcal{O}(\mathcal{C}n^5 \log \kappa)$ bits of private as well as A-cast communication for reaching agreement on $t+1$ bit message. Now simulating the A-cast by the A-cast protocol of [29], we find that the ABA protocol of Chapter 9 requires a private communication of $\mathcal{O}(n^7\kappa)$ bits, to agree on a $t+1$ bit message. But in ACS, we require an ABA protocol that can reach agreement on a single bit. So we use the multi-valued ABA protocol of Chapter 9 in the ACS protocol in the following manner: Every party inputs $t+1$ bits in each ABA_j, such that the first bit is the actual input and the remaining t bits are always 0. This allows us to use the multi-valued ABA protocol of Chapter 9 in the ACS protocol. So our ACS protocol will require a private communication of $\mathcal{O}(|\mathcal{R}|n^7\kappa)$ bits. Thus we have the following theorem:

Theorem 14.20 *Using the multi-valued* $(\rho, 0)$*-ABA protocol of Chapter 9 as block-box, protocol* Patra-ACS *requires private communication of* $\mathcal{O}(|\mathcal{R}|n^7\kappa)$ *bits.*

14.3.2 Approach used in the BA protocol of [75]

Before presenting protocol Optimal-ABA, we briefly recall the approach used in [75] for designing the communication optimal multi-valued BA protocol in synchronous settings. The following description will help to compare and contrast the techniques used in BA of [75] and our ABA protocol. Moreover, this will also help to discern the comparative difficulties in achieving certain tasks in asynchronous network rather than in a synchronous network.

The protocol of [75] requires $n = 2t+1$ parties. So $|\mathcal{P}| = 2t+1$. The BA protocol was structured into three stages: (a) Checking, (b) Consolidation and (c) Claiming Stage. In the Checking Stage, the parties in $\mathcal{P}$ compare their respective messages and jointly determine an accepting subset $\mathcal{P}_{acc} \subseteq \mathcal{P}$ of size at least $n-t$, such that all 'accepting' parties hold the same message, and all (honest) parties holding this message are 'accepting'. This stage can be aborted when inconsistencies among honest parties are detected. If this stage is not aborted then the BA protocol proceeds to Consolidation Stage where the parties in $\mathcal{P}_{acc}$ help to decide on a happy subset $\mathcal{P}_{ok} \subseteq \mathcal{P}$, such that all 'happy' parties hold the same message, and the majority of 'happy' parties are honest. Also this stage may be aborted in case of inconsistencies among the honest parties' inputs. Consolidation Stage is very important and introduces a few new ideas. But a careful checking will reveal that the same ideas can not be implemented in asynchronous network even in the presence of $n = 3t + 1$ parties. That is why we introduce a new sets of ideas in our ABA protocol which is presented in the next section. Finally, if Consolidation Stage is not aborted then BA protocol of [75] proceeds to the last stage called Claiming Stage. In the Claiming Stage, the parties in $\mathcal{P}_{ok}$ distribute their common message to the unhappy parties i.e the parties in $\mathcal{P} \setminus \mathcal{P}_{ok}$. This stage will never be aborted and hence at the end every party will output a common value. If the BA protocol aborts during Checking and Consolidation Stages then every party decides on a predefined default value.

14.3.3 Protocol Optimal-ABA

We may design a naive ABA protocol as follows: Every party is asked to A-cast their input message and then using Patra-ACS the parties agree on a set of $n - t = 2t + 1$ parties, say $\mathcal{T}$, whose A-casts have been terminated; if m^* is the value received from the A-cast of the majority of the parties in $\mathcal{T}$, then everybody agrees on m^*; otherwise everybody agrees on predefined $m^\dagger$. But using our communication optimal protocol Optimal-A-cast, this naive protocol requires a communication of $\mathcal{O}(\ell n^2 + n^8 \kappa)$ bits. So it is challenging to design ABA with communication complexity of $\mathcal{O}(\ell n + poly(n, \kappa))$ bits.

To meet the above challenge, our protocol Optimal-ABA uses the so-called *player-elimination framework*, along with several novel ideas. So far player-elimination [98] has been used *only* in the context of synchronous and asynchronous MPC [98, 52, 14, 143, 135]. Ours is the first non-MPC application of player-elimination. We would refer it by *party-elimination*, rather than player-elimination in our context (as we use the term party in place of player). In the party-elimination framework, the computation of Optimal-ABA is divided into t segments, where in each segment the parties agree on an $\frac{\ell}{t}$ bit message, considering $\frac{\ell}{t}$ bits of their original input as the *input message* of the segment. In particular, the parties divide their original message into t blocks, each of size $\frac{\ell}{t}$ bits and in α^{th} segment $\mathcal{S}_\alpha$, the parties reach agreement on an $\frac{\ell}{t}$ bit message, considering *only* the α^{th} block as the input message. Each segment terminates eventually with the parties having common output of $\frac{\ell}{t}$ bits; moreover if the honest parties start a segment with the same block of $\frac{\ell}{t}$ bits, then they agree on that common input.

The computation of a segment is carried out in a *non-robust* fashion, in the sense that if all the parties including the corrupted parties behave according to the protocol then the segment successfully achieves its task; otherwise the segment may fail in which case it outputs a *triplet of parties* among which *at least one is corrupted*. In the former case, the next segment will be taken up for computation for reaching agreement with next block of $\frac{\ell}{t}$ bits as input. In the later case, the same segment will be repeated among the set of parties after excluding the parties in the triplet and this continues until the segment becomes successful. It is to be noted that though the computations in a segment may be done among a *subset* of parties from $\mathcal{P}$ (as parties in triplet might be eliminated from $\mathcal{P}$), the agreement in the segment is finally attained over all honest parties in $\mathcal{P}$. It is now easy to see that the t segments may fail at most t times *in total* as t is the upper bound on the number of corrupted parties. After t failures, all the corrupted parties will be removed and therefore there will be no more failure.

We denote the input of party P_i by $m_i \in \{0,1\}^\ell$, which is divided into t blocks, with α^{th} block being denoted by $m_{i\alpha}$, for $\alpha = 1, \ldots, t$. At the beginning of our protocol, we initialize two dynamic variables $n' = n$ and $t' = t$ and one dynamic set $\mathcal{P}' = \mathcal{P}$. $\mathcal{P}'$ denotes the *set of non-eliminated* parties and contains n' parties, out of which at most t' can be corrupted. In every segment $\mathcal{S}_\alpha$ the computation is structured into three main phases: (a) **Checking Phase**, (b) **Expansion Phase** and (c) **Output Phase**. The segment failure may occur *only in the second phase* and hence *only* the first two phases of a segment may be repeated several times (bounded by t); once the first two phases are successful for a segment, the segment will always be successfully completed after robustly executing the third phase. So at the end of segment $\mathcal{S}_\alpha$, every honest party will agree on a common $\frac{\ell}{t}$ bits, denoted by m^*_α. Moreover if the honest parties start with common input (i.e $m_{i\alpha}$'s are equal for all honest parties), then m^*_α will be same as that common input.

1. **Checking Phase:** Here the parties, on having private input message of $\frac{\ell}{t}$ bits each (i.e $m_{i\alpha}$'s), jointly perform some computation in order to determine and agree on a set of $t'+1$ parties called $\mathcal{P}'_{ch} \subseteq \mathcal{P}'$, such that the honest parties in $\mathcal{P}'_{ch}$ hold a common ℓ/t bit message, say m^*_α. In case of failure due to the inconsistencies among the inputs of the honest parties, parties abort any further computation for current segment and agree on a predefined message $m^\dagger_\alpha$. So in this case current segment terminates with all honest parties agreeing on common output $m^*_\alpha = m^\dagger_\alpha$. On the other hand, if $\mathcal{P}'_{ch}$ is generated and agreed among the parties, then the computation for current segment proceeds to the next phase. It is to be noted that $\mathcal{P}'_{ch}$ will be *always obtained* if the initial messages of the honest parties in $\mathcal{P}'$ are same.

2. **Expansion Phase:** Here the parties in $\mathcal{P}'_{ch}$ on holding a common message m^*_α help other parties to receive m^*_α. Specifically here the parties jointly perform some computation in conjunction with the parties in $\mathcal{P}'_{ch}$ to expand $\mathcal{P}'_{ch}$ to a set of $2t'+1$ parties, denoted by $\mathcal{P}'_{ex}$ (with $\mathcal{P}'_{ch} \subset \mathcal{P}'_{ex} \subset \mathcal{P}'$) such that all honest parties in $\mathcal{P}'_{ex}$ hold m^*_α. *The expansion technique is the most crucial and novel part of our protocol.* But the computation of this phase is non-robust and hence either one of the following is guaranteed: (a) $\mathcal{P}'_{ex}$ is constructed successfully or (b) a triplet of parties (P_i, P_j, P_k) is obtained, such that at least one of the three parties is corrupted. If the former case happens, then parties proceed to execute **Output Phase**. If the later case happens, then n' and t' are reduced by 3 and 1 respectively and the current segment is repeated from the beginning with updated n' and t' and $\mathcal{P}' = \mathcal{P}' \setminus \{P_i, P_j, P_k\}$. Note that n', t' and $\mathcal{P}'$ always satisfy: $n' = 3t'+1$ and $|\mathcal{P}'| = n'$.

3. **Output Phase:** Here the parties in $\mathcal{P}'_{ex}$ help the parties in $\mathcal{P} \setminus \mathcal{P}'_{ex}$ (**not** $\mathcal{P}' \setminus \mathcal{P}'_{ex}$) to learn the common ℓ/t message m^*_α held by the honest parties in

$\mathcal{P}'_{ex}$. After this phase, current segment terminates with common output m^*_α and the parties proceed to the computation of next segment. The implementation of this phase is very similar to the implementation of Claiming Stage of the BA protocol of [75]. The **Output Phase** of Optimal-A-cast (as presented in previous section) adopts the technique used in Claiming Phase of [75]. Hence **Output Phase** for our ABA is almost identical to the **Output Phase** of Optimal-A-cast and is just the customized version of the **Output Phase** of Optimal-A-cast in the current settings.

Now the overall structure of Optimal-ABA is given in Fig. 14.5.

Figure 14.5: Overall structure of Protocol Optimal-ABA.

Protocol Optimal-ABA($\mathcal{P}, \epsilon, \delta$)

CODE FOR P_i: Every party in $\mathcal{P}$ executes this code.

1. Set $n' = n$, $t' = t$ and $\mathcal{P}' = \mathcal{P}$.
2. Initialize $\alpha = 1$.
3. While $\alpha \leq t$, do the following for segment $\mathcal{S}_\alpha$ with input $m_{i\alpha}$ and with current n', t' and $\mathcal{P}'$ to agree on m^*_α:
   (a) **Checking Phase:** Participate in the code Checking, presented in Fig. 14.6 to determine and agree on $\mathcal{P}'_{ch} \subseteq \mathcal{P}'$ of size $t'+1$ such that all the honest parties in $\mathcal{P}'_{ch}$ hold common $\frac{\ell}{t}$ bits, say m^*_α. If $\mathcal{P}'_{ch}$ is generated then proceed to the next phase. Otherwise set m^*_α to some predefined value $m^\dagger_\alpha \in \{0,1\}^{\frac{\ell}{t}}$, set $\alpha = \alpha + 1$ and terminate the current segment with output m^*_α.
   (b) **Expansion Phase:** Participate in code, Expansion presented in Fig. 14.7 to expand $\mathcal{P}'_{ch}$ to contain $2t'+1$ parties, denoted by $\mathcal{P}'_{ex}$ such that $\mathcal{P}'_{ch} \subset \mathcal{P}'_{ex} \subseteq \mathcal{P}'$ and all honest parties in $\mathcal{P}'_{ex}$ hold m^*_α. If $\mathcal{P}'_{ex}$ is generated successfully then proceed to next phase. Otherwise output a triple (P_m, P_l, P_k), set $n' = n' - 3$, $t' = t' - 1$ and $\mathcal{P}' = \mathcal{P}' \setminus \{P_m, P_l, P_k\}$ and repeat the computation of the current segment.
   (c) **Output Phase:** Participate in code Output presented in Fig. 14.2 and output m^*_α upon termination, set $\alpha = \alpha + 1$ and terminate the current segment.
4. Output m^* which is the concatenation of $m^*_1, \ldots, m^*_t$ and terminate the protocol.

In the sequel, we will pursue an in-depth discussion on the implementation and properties of each of the above three phases.

14.3.3.1 Checking Phase

The following description outlines the idea used for this phase. The aim of this phase is to either agree on a set $\mathcal{P}'_{ch}$ of size $t'+1$ such that all the honest parties in

$\mathcal{P}'_{ch}$ hold common message, say m^*_α, or decide that such set may not exist. When all the honest parties start with same input message then $\mathcal{P}'_{ch}$ can be always found out and agreed upon. To achieve the above task, every party Bracha-A-casts a (random key, hash value) pair corresponding to his message. The parties then agree on a set $\mathcal{I}$ of $n' - t'$ parties whose Bracha-A-cast will be eventually received by every honest party. This can be achieved using one instance of Patra-ACS.

Now every party P_i prepares a response vector $\overrightarrow{v_i}$, indicating whether the hash value of every $P_j \in \mathcal{I}$ is indeed the hash value of his own message $m_{i\alpha}$ with respect to P_j's hash key (this should ideally be the case, when P_i and P_j are honest and their input messages are identical, i.e $m_{i\alpha} = m_{j\alpha}$). P_i Bracha-A-casts $\overrightarrow{v_i}$. Now the parties again agree on a set of $n' - t'$ parties, say $\mathcal{J}$ whose Bracha-A-cast with their $\overrightarrow{v_i}$ has been terminated. *Now notice that if all honest parties start with common input, then the vectors of the honest parties in $\mathcal{J}$ would be identical and would have at least $t' + 1$ 1's at the locations corresponding to the $t' + 1$ honest parties in $\mathcal{I}$*. So now the parties try to find a set of at least $t' + 1$ parties in $\mathcal{J}$, whose vectors are identical and have *at least $t' + 1$ 1's in them*. If found, then any subset of $t' + 1$ parties from that set (say $t' + 1$ parties with smallest index) will be considered as $\mathcal{P}'_{ch}$. It is easy to show that $\mathcal{P}'_{ch}$ will be *obtained always* if the initial messages of the honest parties in $\mathcal{P}'$ are same. Moreover it can also be shown that the honest parties in $\mathcal{P}'_{ch}$ hold common message, say m^*_α with very high probability (see Lemma 14.22). But if $\mathcal{P}'_{ch}$ is not found, then the honest parties know that their input messages are inconsistent and hence they agree that such set can not be found. The steps performed so far are enough for our current phase.

But we need to do some more task for the requirement of next phase i.e **Expansion Phase**. In **Expansion Phase**, we require that every honest party in $\mathcal{P}'$ should hold a distinct random hash key and hash value of the message corresponding to every party in $\mathcal{I}$, such that for every $P_i \in \mathcal{I}$ and $P_j \in \mathcal{P}'$ the hash key and hash value that P_j has received from P_i should not be known to anybody other than P_i and P_j. Though achieving this in synchronous network is easy, it needs some amount of effort in asynchronous network. We do this in the following way:

> Party $P_i \in \mathcal{P}'$ selects n' random hash keys from $\mathbb{F}$ (one corresponding to every party in $\mathcal{P}'$) and commits j^{th} (key, hash value) pair of his message $m_{i\alpha}$ using two instances of SAVSS-Share, apart from Bracha-A-casting a (random key, hash value) pair corresponding to his message. Now the parties agree on a set of $n' - t'$ parties, say $\mathcal{I}$, whose instance of Bracha-A-cast as well as n instances of SAVSS-Share has been terminated. Now the i^{th} (key, hash value) pair of every $P_j \in \mathcal{I}$ is P_i-private-reconstructed *only* by $P_i \in \mathcal{P}'$, using SAVSS-Rec-Private. This ensures that every (honest) party $P_i \in \mathcal{P}'$ receives i^{th} (key, hash value) pair of *every* $P_j \in \mathcal{I}$, with the guarantee that the pair is known *only to P_i and P_j*. Moreover now the parties continue the tasks for current phase with the Bracha-A-casted information of the parties in $\mathcal{I}$ in the same way as discussed before.

The code that implements this phase is given in Fig. 14.6.

Before proving the properties of code Checking, we define the following event:

Figure 14.6: Code for Checking Phase.

Checking

To avoid notational clutter, we assume that $\mathcal{P}'$ is the set of first n' parties

Code for $P_i \in \mathcal{P}'$: Every party in $\mathcal{P}'$ executes this code

1. On having input $m_{i\alpha}$,
   (a) choose a random hash key r_i from $\mathbb{F}$ and Bracha-A-cast $(r_i, \mathcal{V}_i)$ where $\mathcal{V}_i = \mathcal{U}_\kappa(m_{i\alpha}, r_i)$;
   (b) choose n' random hash keys $r_{i1}, \ldots, r_{in'}$ from $\mathbb{F}$ and commit $(r_{ij}, \mathcal{V}_{ij})$ where $\mathcal{V}_{ij} = \mathcal{U}_\kappa(m_{i\alpha}, r_{ij})$, by executing SAVSS-Share$(P_i, \mathcal{P}', r_{ij}, \frac{\epsilon}{n^2})$ and SAVSS-Share$(P_i, \mathcal{P}', \mathcal{V}_{ij}, \frac{\epsilon}{n^2})$.
2. Participate in SAVSS-Share$(P_j, \mathcal{P}', r_{jk}, \frac{\epsilon}{n^2})$ and SAVSS-Share$(P_j, \mathcal{P}', \mathcal{V}_{jk}, \frac{\epsilon}{n^2})$ for every $P_j \in \mathcal{P}'$ and $k = 1, \ldots, n'$.
3. Participate in Patra-ACS$(\mathcal{P}', n' - t', \frac{\epsilon}{n})$ to agree on a set of $n' - t'$ parties from $\mathcal{P}'$, denoted as $\mathcal{I}$, whose instance of Bracha-A-cast as well as all the $2n'$ instances of SAVSS-Share will be eventually terminated (by all honest parties in $\mathcal{P}'$).
4. Wait to receive $(r_j, \mathcal{V}_j)$ from the Bracha-A-cast of every $P_j \in \mathcal{I}$.
5. Wait to terminate all $2n'$ instances of SAVSS-Share of every party in $\mathcal{I}$. Participate in SAVSS-Rec-Private$(P_j, \mathcal{P}', r_{jk}, P_k, \frac{\epsilon}{n^2})$ and SAVSS-Rec-Private$(P_j, \mathcal{P}', \mathcal{V}_{jk}, P_k, \frac{\epsilon}{n^2})$ for every $P_j \in \mathcal{I}$ and every $P_k \in \mathcal{P}'$ for P_k-private-reconstruction of $(r_{jk}, \mathcal{V}_{jk})$.
6. Obtain $(r_{ji}, \mathcal{V}_{ji})$ pair from SAVSS-Rec-Private$(P_j, \mathcal{P}', r_{ji}, P_i, \frac{\epsilon}{n^2})$ and SAVSS-Rec-Private$(P_j, \mathcal{P}', \mathcal{V}_{ji}, P_i, \frac{\epsilon}{n^2})$ corresponding to every $P_j \in \mathcal{I}$.
7. Construct n length vector $\overrightarrow{v_i}$, where $\overrightarrow{v_i}[j] = \begin{cases} \perp & \text{If } P_j \notin \mathcal{I} \\ 1 & \text{If } P_j \in \mathcal{I} \text{ and } \mathcal{V}_j = \mathcal{U}_\kappa(m_{i\alpha}, r_j). \\ 0 & \text{If } P_j \in \mathcal{I} \text{ and } \mathcal{V}_j \neq \mathcal{U}_\kappa(m_{i\alpha}, r_j). \end{cases}$ Bracha-A-cast $\overrightarrow{v_i}$.
8. Participate in Patra-ACS$(\mathcal{P}', n' - t', \frac{\epsilon}{n})$ to agree on a set of $n' - t'$ parties from $\mathcal{P}'$, denoted as $\mathcal{J}$, whose Bracha-A-cast with an n length vector has been terminated.
9. Check whether there is a unique set of at least $t' + 1$ parties in $\mathcal{J}$ such that their vectors are identical and have at least $t' + 1$ 1's in them (Note that this can be done easily in polynomial time).
   (a) If yes, then let $\mathcal{P}'_{ch}$ be the set containing exactly $t' + 1$ parties (say the parties with first $t' + 1$ smallest indices) out of those parties. Let $\overrightarrow{v}$ be an n length vector, where $\overrightarrow{v}[i] = 1$ if the i^{th} location of the vectors of all parties in $\mathcal{P}'_{ch}$ is 1, otherwise $\overrightarrow{v}[i] = \perp$. Moreover, let $\mathcal{I}_1 = \{P_i \in \mathcal{I} \text{ such that } \overrightarrow{v}[i] = 1\}$. Assign $m^*_\alpha = m_{i\alpha}$ if $P_i \in \mathcal{P}'_{ch}$.
   (b) If not, then decide that $\mathcal{P}'_{ch}$ can not be found.

Event E**:** *Let E be an event in an execution of* Checking*, defined as follows: All invocations of AVSS scheme initiated by the parties in $\mathcal{I}$ have been terminated with correct output. More clearly, E means that all the invocations of AVSS protocols initiated by the parties in $\mathcal{I}$ will satisfy termination property and correctness property. It is easy to see that event E occurs with probability at least $(1 - |\mathcal{I}|\frac{\epsilon}{n^2}) \approx (1 - \frac{\epsilon}{n})$, as $|\mathcal{I}| = \Theta(n)$ and each instance of the AVSS is executed with error parameter $\frac{\epsilon}{n^2}$.* □

In the sequel, all the lemmas for all the three phases are proved conditioned on event E. Now before presenting our proofs, we discuss the way the proofs are presented. For every phase, we first find the error probability of termination and then find the error probability with which the phase will output its' desired result (i.e correctness of the phase), conditioned on the event that the phase terminates.

Lemma 14.21 (Termination of the Checking Phase) *In a segment $\mathcal{S}_\alpha$, any particular execution of* **Checking Phase** *will be terminated, except with probability $\frac{\epsilon}{n}$, where termination means the code either outputs a set $\mathcal{P}'_{ch}$ of size $t' + 1$ or decide that such set can not be constructed.*

PROOF: Conditioned on event E, an execution of **Checking Phase** will always terminate if both the executions of Patra-ACS terminates and all the instances of Bracha-A-cast terminate. Since Bracha-A-cast has no error in termination and each execution of Patra-ACS terminates, except with error probability $\frac{\epsilon}{n}$, an execution of **Checking Phase** will terminate, except with probability $\frac{\epsilon}{n}$. □

Lemma 14.22 (Correctness of the Checking Phase) *In any particular execution of* **Checking Phase** *in a segment $\mathcal{S}_\alpha$, the honest parties in $\mathcal{P}'_{ch}$ (if it is found) hold a common message m^*_α, except with error probability of at most $\frac{\delta}{n^2}$. Moreover, if the honest parties start $\mathcal{S}_\alpha$ with common message m_α, then $\mathcal{P}'_{ch}$ will always be found with $m^*_\alpha = m_\alpha$.*

PROOF: We prove the first part of the lemma. If $\mathcal{P}'_{ch}$ contains exactly one honest party, then first part is trivially true with m^*_α being the input message of the sole honest party in $\mathcal{P}'_{ch}$. So let $\mathcal{P}'_{ch}$ contain at least two honest parties. We now show that the messages of every pair of honest parties (P_i, P_j) in $\mathcal{P}'_{ch}$ are same. Recall that the response vectors $\overrightarrow{v_i}$ and $\overrightarrow{v_j}$ of P_i and P_j are identical and have at least $t + 1$ 1's in them. Moreover, $\mathcal{I}_1$ contains all P_k's such that $\overrightarrow{v_i}[k] = \overrightarrow{v_j}[k] = 1$. Evidently, $|\mathcal{I}_1| \geq t+1$. So there is at least one *honest* party in $\mathcal{I}_1$, say P_k, such that $\overrightarrow{v_i}[k] = \overrightarrow{v_j}[k] = 1$. This implies that $\mathcal{V}_k = \mathcal{U}_\kappa(m_{i\alpha}, r_k)$ and $\mathcal{V}_k = \mathcal{U}_\kappa(m_{j\alpha}, r_k)$ holds for P_i and P_j respectively, where P_i has received $(\mathcal{V}_k, r_k)$ from P_k (by Bracha-A-cast) and P_j has received $(\mathcal{V}_k, r_k)$ from P_k (by Bracha-A-cast). Now by **Collision Theorem** (see Theorem 14.5), it easily follows that $m_{i\alpha} = m_{k\alpha}$ and $m_{j\alpha} = m_{k\alpha}$, except with probability at most $\frac{\delta}{n^3}$ (replacing ℓ by $\frac{\ell}{n}$ in Theorem 14.5). Consequently $m_{i\alpha} = m_{j\alpha}$, except with probability at most $\frac{\delta}{n^3}$.

Now let us fix an honest party, say P_i in $\mathcal{P}'_{ch}$. If P_i's value is equal to every honest P_j's value in $\mathcal{P}'_{ch}$, then it means that all honest parties in $\mathcal{P}'_{ch}$ hold a common message m^*_α. This happens except with error probability $|H|\frac{\delta}{n^3}$, where H is the set of honest parties in $\mathcal{P}'_{ch}$. As $|H|$ can be $\mathcal{O}(t)$, we have $|H|\frac{\delta}{n^3} \approx \frac{\delta}{n^2}$.

We now prove the second part. When all honest parties start with same input m_α, the vectors of all honest parties in $\mathcal{J}$ will have 1 at the locations corresponding to the honest parties in $\mathcal{I}$. Since there are at least $t' + 1$ honest parties in both $\mathcal{I}$ and $\mathcal{J}$, $\mathcal{P}'_{ch}$ can always be found and now it is easy to see that all honest parties in $\mathcal{P}'_{ch}$ will hold m_α. □

14.3.3.2 Expansion Phase

If $\mathcal{P}'_{ch}$ is found and agreed upon in the previous phase, then the parties proceed to expand $\mathcal{P}'_{ch}$ in order to obtain $\mathcal{P}'_{ex}$. For that we first initiate $\mathcal{K} = \mathcal{P}'_{ch}$ and $\overline{\mathcal{K}} = \mathcal{P}' \setminus \mathcal{K}$. Then $\mathcal{K}$ will be expanded to contain $2t'+1$ parties and we will assign $\mathcal{K}$ to $\mathcal{P}'_{ex}$ when $\mathcal{K}$ contains $2t'+1$ parties. We call the $\mathcal{K}$ containing $t'+1$ parties as 'initial' $\mathcal{K}$ and likewise the $\mathcal{K}$ containing $2t'+1$ parties as 'final' $\mathcal{K}$. The expansion (transition from 'initial' $\mathcal{K}$ to 'final' $\mathcal{K}$) takes place in a sequence of t' iterations. In each iteration, either $\mathcal{K}$ is expanded by one by including a new party or in case of failure a conflict triplet is returned. In the later case, the current segment fails and hence it is again repeated with renewed value of n', t' and $\mathcal{P}'$ (i.e after excluding the parties in the triplet from $\mathcal{P}'$).

So this phase starts as follows: First an injective mapping $\varphi : \mathcal{K} \rightarrow \overline{\mathcal{K}}$ is defined. Now a party $P_i \in \mathcal{K}$ sends his message m^*_α to party $\varphi(P_i) \in \overline{\mathcal{K}}$. A party $P_i \in \overline{\mathcal{K}}$ on receiving a message m^*_α from $\varphi^{-1}(P_i) \in \mathcal{K}$, calculates vector $\overrightarrow{v_i}$ with the (key, hash value) pair of the parties only in $\mathcal{I}_1$ (at all other locations $\perp$ is placed) and with m^*_α as the message. P_i then Bracha-A-casts `Matched-`P_i if $\overrightarrow{v_i}$ is identical to $\overrightarrow{v}$ (which was calculated in Checking). Otherwise let k be the minimum index in $\overrightarrow{v_i}$ such that $\overrightarrow{v_i}[k] \neq \overrightarrow{v}[k]$, then P_i Bracha-A-casts a conflict triplet $(\varphi^{-1}(P_i), P_i, P_k)$. Clearly, one of the three parties in the triplet must be corrupted. The parties now invoke an instance of Patra-ACS to agree on a single party, say P_l from $\overline{\mathcal{K}}$ whose Bracha-A-cast has been terminated. Such a party from $\overline{\mathcal{K}}$ can always be found as there exists at least one honest $P_m \in \mathcal{K}$ which will be mapped to another honest $P_l = \varphi(P_m) \in \overline{\mathcal{K}}$ and P_l will eventually receive m^*_α from P_m and successfully Bracha-A-cast some message (see Lemma 14.24).

Now there are two cases. If $(\varphi^{-1}(P_l), P_l, P_k)$ is received from the Bracha-A-cast of P_l, then the computation stops here and the triplet $(\varphi^{-1}(P_l), P_l, P_k)$ is returned. If `Matched-`P_l is received from the Bracha-A-cast of P_l, then P_l is included in $\mathcal{K}$ and excluded from $\overline{\mathcal{K}}$. P_l now finds a unique party from the set of parties in $\overline{\mathcal{K}}$ that are never mapped before (say the unmapped party with smallest index) and sends m^*_α to it. Again the party who receives the message, calculates response vector with the received message and Bracha-A-casts either a conflict triplet or `Matched` signal. Then parties invokes an instance of Patra-ACS to agree on a single party from $\overline{\mathcal{K}}$ whose Bracha-A-cast has been terminated and this process continues until either $|\mathcal{K}|$ becomes $2t'+1$ or the segment is failed with some triplet in some iteration. Though it is non-intuitive that in every iteration the parties will be able to agree on a single party from $\overline{\mathcal{K}}$ by executing Patra-ACS, this will indeed happen and we prove this clearly in Lemma 14.24. If $\mathcal{K}$ becomes of size $2t'+1$, it is assigned to $\mathcal{P}'_{ex}$. The code for this phase is given in Fig. 14.7.

We now prove the properties of **Expansion Phase**.

Lemma 14.23 *In a segment $\mathcal{S}_\alpha$, in any iteration of* `while` *loop (in an execution of* **Expansion Phase***), no two different parties in $\mathcal{K}$ are mapped to the same party in $\overline{\mathcal{K}}$. Also in case* `while` *loop is completed with $\mathcal{K}$ containing $2t'+1$ parties, only the last entrant in 'final' $\mathcal{K}$ is not mapped to any party.*

Proof: From the protocol steps, it is clear that a party in $\mathcal{K}$ is mapped only once. Now we show that no pair (P_i, P_j) in $\mathcal{K}$ is mapped to same party. This

Figure 14.7: Code for the Expansion Phase.

Expansion

Code for $P_i \in \mathcal{P}'$: Every party in $\mathcal{P}'$ executes this code

1. Assign $\mathcal{K} = \mathcal{P}'_{ch}$ and $\overline{\mathcal{K}} = \mathcal{P}' \setminus \mathcal{K}$.

2. Define an injective mapping $\varphi : \mathcal{K} \rightarrow \overline{\mathcal{K}}$ where $\overline{\mathcal{K}} = \mathcal{P}' \setminus \mathcal{K}$ as follows: the party with smallest index in $\mathcal{K}$ is associated with the party with smallest index in $\overline{\mathcal{K}}$. Let $\mathcal{M} = \varphi(\mathcal{K})$ ($\subset \overline{\mathcal{K}}$, as $|\mathcal{K}|$ is exactly $t' + 1$) be the set of currently *mapped* partied in $\overline{\mathcal{K}}$. Let $\overline{\mathcal{M}} = \overline{\mathcal{K}} \setminus \mathcal{M}$ be the set of currently *unmapped* partied in $\overline{\mathcal{K}}$.

3. If $P_i \in \mathcal{K}$, then send m^*_α to $\varphi(P_i)$.

4. If $P_i \in \overline{\mathcal{K}}$ and has received message m^*_α from $\varphi^{-1}(P_i) \in \mathcal{K}$, then calculate vector $\overrightarrow{v_i}$ of length n as follows: $\overrightarrow{v_i}[j] = \begin{cases} \perp & \text{If } P_j \notin \mathcal{I}_1 \\ 1 & \text{If } P_j \in \mathcal{I}_1 \text{ and } \mathcal{V}_{ji} = \mathcal{U}_\kappa(m^*_\alpha, r_{ji}). \\ 0 & \text{If } P_j \in \mathcal{I}_1 \text{ and } \mathcal{V}_{ji} \neq \mathcal{U}_\kappa(m_\alpha, r_{ji}). \end{cases}$ Recall that $(r_{ji}, \mathcal{V}_{ji})$ pair was obtained by P_i in Checking from SAVSS-Rec-Private$(P_j, \mathcal{P}', r_{ji}, P_i, \frac{\epsilon}{n^2})$ and SAVSS-Rec-Private$(P_j, \mathcal{P}', \mathcal{V}_{ji}, P_i, \frac{\epsilon}{n^2})$. If $\overrightarrow{v_i}$ is identical to $\overrightarrow{v}$ then Bracha-A-cast `Matched-`P_i; otherwise let k be the minimum index in $\overrightarrow{v_i}$ such that $\overrightarrow{v_i}[k] \neq \overrightarrow{v}[k]$, then Bracha-A-cast (P_j, P_i, P_k), where $P_j = \varphi^{-1}(P_i)$.

5. `while` $|\mathcal{K}| < 2t' + 1$ `do`:

   (a) Participate in an instance of Patra-ACS$(\mathcal{M}, 1, \frac{\epsilon}{n^2})$ to agree on a single party from $\mathcal{M}$ whose Bracha-A-cast has been terminated. Let the party be P_l.

   (b) If (P_m, P_l, P_k) is received from Bracha-A-cast of P_l, then stop any further computation and output the triplet (P_m, P_l, P_k).

   (c) If `Matched-`P_l is received from Bracha-A-cast of P_l, then set $\mathcal{K} = \mathcal{K} \cup \{P_l\}$, $\overline{\mathcal{K}} = \overline{\mathcal{K}} \setminus \{P_l\}$ and $\mathcal{M} = \mathcal{M} \setminus \{P_l\}$.

   (d) Define a mapping, which maps P_l to the party in $\overline{\mathcal{M}}$ with the smallest index, say P_m. Set $\overline{\mathcal{M}} = \overline{\mathcal{M}} \setminus \{P_m\}$ and $\mathcal{M} = \mathcal{M} \cup \{P_m\}$.

   (e) If $P_i = P_l$, then send m^*_α to P_m.

   (f) If $P_i = P_m$ and P_i has received message m^*_α from P_l, then calculate vector $\overrightarrow{v_i}$ of length n in the same way as in step 4. If $\overrightarrow{v_i}$ is identical to $\overrightarrow{v}$ then Bracha-A-cast `Matched-`P_i; otherwise let k be the minimum index in $\overrightarrow{v_i}$ such that $\overrightarrow{v_i}[k] \neq \overrightarrow{v}[k]$, then Bracha-A-cast (P_l, P_i, P_k).

6. Set $\mathcal{P}'_{ex} = \mathcal{K}$. If $P_i \in \mathcal{P}'_{ex}$, then consider m^*_α as the final message.

is true as φ is injective and also every time a party P_i from $\mathcal{K}$ is mapped to a party P_k in $\overline{\mathcal{M}}$ (set of unmapped parties), P_k is never mapped again as it is immediately transferred to $\mathcal{M}$ (set of mapped parties).

Now we show that there will be enough number of parties in $\overline{\mathcal{M}}$ to be mapped in all iterations, except the last one. We consider the worse case, where the while loop is executed completely for t' iterations (as 'initial' $|\mathcal{K}|$ is $t'+1$ and t' more parties have to enter to make 'final' $\mathcal{K}$ of size $2t'+1$) without outputting a triplet. Now as per the protocol, at the beginning of the `while` loop, $\mathcal{K} = t'+1$, $\overline{\mathcal{K}} = 2t'$, $\mathcal{M} = t'+1$ and $\overline{\mathcal{M}} = 2t' - (t'+1) = t'-1$. In i^{th} iteration, a party, say P_l from $\mathcal{M}$ (hence from $\overline{\mathcal{K}}$) enters into $\mathcal{K}$ and gets mapped to an unmapped party in $\overline{\mathcal{M}}$ (hence in $\overline{\mathcal{K}}$). As a result: (a) $|\mathcal{K}|$ increases by 1, (b) $|\overline{\mathcal{K}}|$ decreases by 1, (c) $|\mathcal{M}|$ remains same and (d) $|\overline{\mathcal{M}}|$ decreases by 1. So after $t'-1$ iterations, the following hold: (a) $|\mathcal{K}| = 2t'$, (b) $|\overline{\mathcal{K}}| = t'+1$, (c) $|\mathcal{M}| = t'+1$ and (d) $|\overline{\mathcal{M}}| = 0$. Hence only after the mapping done in $(t'-1)^{th}$ iteration, $\overline{\mathcal{M}}$ becomes empty. In the last iteration (t'^{th}), another party from $\mathcal{M}$ (hence from $\overline{\mathcal{K}}$) is finally included in $\mathcal{K}$ which need not be mapped to any more party as $\mathcal{K}$ becomes exactly $2t'+1$ here. □

Lemma 14.24 *In a particular execution of* **Expansion Phase** *in a segment $\mathcal{S}_\alpha$, $|\mathcal{K}|$ will increase by one with probability $(1-\frac{\epsilon}{n^2})$, in every iteration of `while` loop until the `while` loop is completed due to $|\mathcal{K}| = 2t'+1$ or broken due to the output of triplet.*

PROOF: To prove the lemma, we show that in every iteration of the `while` loop, the parties will be able to agree on a single party (using Patra-ACS) from $\overline{\mathcal{K}}$ (thus from $\mathcal{M}$) (except with probability $\frac{\epsilon}{n^2}$, as the instance of Patra-ACS may not terminate with probability $\frac{\epsilon}{n^2}$), whose Bracha-A-cast will be terminated. In other words, we assert that in every iteration of the `while` loop, there will exist one party from $\overline{\mathcal{K}}$ (and thus from) who will eventually Bracha-A-cast a response. Moreover, this will be true, until the `while` loop is either over or broken due to the output of triplet. For this, we claim that in every iteration of `while` loop, there must be an honest party, say P_i, belonging to $\mathcal{K}$, such that P_i is mapped to another honest party, say P_j, belonging to $\overline{\mathcal{K}}$. Moreover, honest P_i's message will eventually reach to honest P_j, who will then Bracha-A-cast his response, which is either an n length vector or triplet of parties.

At the time of entering into the loop for the first time, let among $t'+1$ parties in $\mathcal{K}$ there are $0 \leq c \leq t'$ corrupted parties. So the remaining $t'-c$ corrupted parties are in 'initial' $\overline{\mathcal{K}}$. In worst case, c corrupted parties and $t'-c$ honest parties from $\mathcal{K}$ may be mapped to c honest parties and $t'-c$ corrupted parties, respectively from $\overline{\mathcal{K}}$. Still $\mathcal{K}$ contains at least one honest party which is bound to be mapped to another honest party from $\overline{\mathcal{K}}$, as there is no other unmapped corrupted party in $\overline{\mathcal{K}}$. So our claim holds for first iteration. In general in i^{th} iteration, there are $t'+i$ parties in $\mathcal{K}$ out of which say c with $0 \leq c \leq t'$ are corrupted parties. So extending the previous argument for this general case, there are i honest parties in $\mathcal{K}$ who are mapped to i honest parties in $\overline{\mathcal{K}}$. Among these i mappings, $i-1$ might correspond to previous $i-1$ iterations. But still one mapping is left for i^{th} iteration. Now let the mapping be from honest $P_j \in \mathcal{K}$ to honest $P_k \in \overline{\mathcal{K}}$.

So P_j's message reaches to P_k eventually and P_k tries to prepare $\overrightarrow{v_k}$ with received message and the (key, hash value) of the parties in $\mathcal{I}_1$. Conditioned on event E, P_k will receive the (key, hash value) of the parties in $\mathcal{I}_1$. Once P_k prepares his vector, he Bracha-A-casts his response (which could be either signal `Matched-`P_k, if $\overrightarrow{v_k} = \overrightarrow{v}$ or a triplet of parties if $\overrightarrow{v_k} \neq \overrightarrow{v}$). If P_k's response is `Matched-`P_k, then $|\mathcal{K}|$ will be incremented by 1; otherwise, the loop will be broken due to the output of a triplet. Hence the lemma. □

Lemma 14.25 (Termination of the Expansion Phase) *In a segment $\mathcal{S}_\alpha$, any particular execution of* **Expansion Phase** *will terminate, except with error probability $\frac{\epsilon}{n}$, where termination means the code either outputs a triplet or a set $\mathcal{P}'_{ex}$ of size $2t'+1$.*

PROOF: From Lemma 14.24, in every iteration of the `while` loop, there will exist one party from $\overline{\mathcal{K}}$ who will eventually Bracha-A-cast a response. Now conditioned on event E, the termination of an execution of **Expansion Phase** depends on the termination of the invoked Patra-ACS protocols and the Bracha-A-casts. Bracha-A-cast has no error in termination. An instance of Patra-ACS terminates, except with probability $\frac{\epsilon}{n^2}$. Since in an execution of **Expansion Phase**, there can be at most t' invocations of Patra-ACS (corresponding to t' iterations of while loop), all of them will terminate, except with probability $t'\frac{\epsilon}{n^2} \approx \frac{\epsilon}{n}$ (since t' can be $\mathcal{O}(t)$). Therefore, an execution of **Expansion Phase** terminates, except with probability $\frac{\epsilon}{n}$. □

Lemma 14.26 (Correctness-I of Expansion Phase) *In a particular execution of* **Expansion Phase** *in a segment $\mathcal{S}_\alpha$, all the honest parties in $\mathcal{P}'_{ex}$ (if found) will hold a common message m^*_α, which was also the common message held by the honest parties in $\mathcal{P}'_{ch}$, except with probability $\frac{\delta}{n^2}$. Moreover if the honest parties start $\mathcal{S}_\alpha$ with same input message m_α, then $m^*_\alpha = m_\alpha$.*

PROOF: Let E_{ex} be the event that all the honest parties in $\mathcal{P}'_{ex}$ hold a common message m^*_α. We have to show that $E_{ex} = (1-\frac{\delta}{n^2})$. Let E_{ch} and $E_{ex\setminus ch}$ be two events defined as follows: E_{ch}: *All honest parties in $\mathcal{P}'_{ch}$ and hence in 'initial' $\mathcal{K}$ hold a common message m^*_α*; and $E_{ex\setminus ch}$: *All honest parties in $\mathcal{P}'_{ex} \setminus \mathcal{P}'_{ch}$ hold a common message m^*_α*. We now assert that $Prob(E_{ch}) = (1-\frac{\delta}{n^2})$ and $Prob(E_{ex\setminus ch}|E_{ch}) = (1-\frac{\delta}{n^2})$.

1. $Prob(E_{ch}) = (1-\frac{\delta}{n^2})$: Follows from Lemma 14.22.

2. $Prob(E_{ex\setminus ch}|E_{ch}) = (1-\frac{\delta}{n^2})$: Let us consider party P_f, who is the first *honest* party to enter into 'initial' $\mathcal{K}$ during **Expansion phase**. Recall that P_f enters into $\mathcal{K}$ (hence $\mathcal{P}'_{ex}$) when it receives a message $\overline{m^*_\alpha}$ from some already existing (possibly corrupted) party P_j in $\mathcal{K}$ and P_f's generated $\overrightarrow{v_f}$ is identical to $\overrightarrow{v}$. We claim that $\overline{m^*_\alpha} = m^*_\alpha$ with probability $(1-\frac{\delta}{n^3})$. For this consider an honest $P_k \in \mathcal{K}$ and an honest P_l in $\mathcal{I}_1$ with $\overrightarrow{v}[l] = 1$ (there is at least one such honest P_l as $|\mathcal{I}_1| \geq t'+1$). By **Collision Theorem**, $m_{k\alpha} = m_{l\alpha} = m^*_\alpha$ with probability at least $(1-\frac{\delta}{n^3})$. Now since $\overrightarrow{v_f} = \overrightarrow{v}$, it implies that $\overrightarrow{v}_f[l] = 1$, as $\overrightarrow{v}[l] = 1$. This further implies that $\overline{m^*_\alpha} = m_{l\alpha}$ with probability at least $(1-\frac{\delta}{n^3})$. This clearly implies that $\overline{m^*_\alpha} = m^*_\alpha$ holds, with probability at least $(1-\frac{\delta}{n^3})$. This is because the key and hash value pair $(r_{lf}, \mathcal{V}_{lf})$ is not known to anyone (including possibly corrupted P_j) other than P_f and P_l. Hence with probability $(1-\frac{\delta}{n^3})$, P_f has received m^*_α from P_j.

   Now let P_s be the second honest party to enter into 'initial' $\mathcal{K}$ during **Expansion phase**. P_s may receive its message either from P_f or from any party belonging to 'initial' $\mathcal{K}$. If P_s receives the message from any party belonging to 'initial' $\mathcal{K}$, then using similar arguments as above, we can show that its message will be m^*_α, except with error probability $\frac{\delta}{n^3}$. On the other hand, if P_s receives the message from P_f, then also its message will be m^*_α, except with error probability $\frac{\delta}{n^3}$.

In general, if an honest party P_i enters into 'initial' $\mathcal{K}$ at sometime, then its message will be equal to m^*_α, except with error probability $(1-\frac{\delta}{n^3})$. As there can be $\Theta(t)$ honest parties to enter in this manner, all the honest parties in $\mathcal{P}'_{ex} \setminus \mathcal{P}'_{ch}$ hold a common message m^*_α, except with error probability $\Theta(t)\frac{\delta}{n^3} \approx \frac{\delta}{n^2}$. Hence $Prob(E_{ex\setminus ch}|E_{ch}) = (1-\frac{\delta}{n^2})$.

Now we have,

$$\begin{aligned}
Prob(E_{ex}) &= Prob(E_{ch} \cap E_{ex\setminus ch}) \\
&= Prob(E_{ch}) \cdot Prob(E_{ex\setminus ch}|E_{ch}) \\
&= (1-\frac{\delta}{n^2}) \cdot (1-\frac{\delta}{n^2}) \\
&= (1-\frac{\delta}{n^2})^2 \geq (1-2\frac{\delta}{n^2}) \approx (1-\frac{\delta}{n^2}).
\end{aligned}$$

Hence the lemma. □

Lemma 14.27 (Correctness-II of Expansion Phase) *In a particular execution of* **Expansion Phase** *in a segment $\mathcal{S}_\alpha$, if a triplet (P_m, P_l, P_k) is* Bracha-A-cast*ed by P_l then at least one of P_m, P_l and P_k is corrupted, except with error probability $\frac{\delta}{n^3}$ where $P_m \in \mathcal{K}, P_l \in \overline{\mathcal{K}}$ and $P_k \in \mathcal{I}_1$.*

PROOF: Let P_m, P_l and P_k are honest, where $P_m \in \mathcal{K}, P_l \in \overline{\mathcal{K}}$ and $P_k \in \mathcal{I}_1$. Since $P_k \in \mathcal{I}_1$, it implies that $\overrightarrow{v}(k) = 1$ holds. Also $P_m \in \mathcal{K}$ implies that $\overrightarrow{v}_m(k) = 1$. This further implies that m^*_α held by P_m is same as $m_{k\alpha}$ held by P_k, except with error probability $\frac{\delta}{n^3}$ (see **Collision Theorem**). Now during **Expansion phase**, P_m sends his m^*_α to P_l and P_l computes $\overrightarrow{v}_l$ with respect to the received m^*_α and the pairs $(r_{jl}, \mathcal{V}_{jl})$, corresponding to every $P_j \in \mathcal{I}_1$. On computing $\overrightarrow{v}_l$, party P_l will find that $\overrightarrow{v}_l(k) = \overrightarrow{v}(k)$, except with error probability $\frac{\delta}{n^3}$. This is because P_k is honest and hence $\mathcal{V}_{kl}$ is the hash value of $m_{k\alpha}$, with respect to the hash key r_{kl}. However, as shown above, m^*_α received by P_l from P_m is same as $m_{k\alpha}$, except with error probability $\frac{\delta}{n^3}$. So except with error probability $\frac{\delta}{n^3}$, P_l will find that $\mathcal{V}_{kl} = \mathcal{U}_\kappa(m^*_\alpha, r_{kl})$. So except with error probability $\frac{\delta}{n^3}$, P_l will not Bracha-A-cast the triplet (P_m, P_l, P_k). So if at all P_l Bracha-A-cast the triplet (P_m, P_l, P_k), then except with probability $\frac{\delta}{n^3}$, at least one of P_m, P_l and P_k is corrupted. □

14.3.3.3 Output Phase

Once the parties agree on $\mathcal{P}'_{ex}$, with all honest parties in it holding some common m^*_α, we need to ensure that m^*_α propagates to all (honest) parties in $\overline{\mathcal{P}_{ex}} = \mathcal{P} \setminus \mathcal{P}'_{ex}$, in order to reach agreement on m^*_α. This is achieved in code Output (presented in Fig. 14.8) with the help of the parties in $\mathcal{P}'_{ex}$. A simple solution could be to ask each party in $\mathcal{P}'_{ex}$ to send his m^*_α to all the parties in $\overline{\mathcal{P}_{ex}}$, who can wait to receive $t'+1$ same m^*_α and then accept m^*_α as the message. This solution will work as there are at least $t'+1$ honest parties in $\mathcal{P}'_{ex}$. But clearly, this requires a communication complexity of $\mathcal{O}(\ell n)$ bits for each segment (and thus $\mathcal{O}(\ell n^2)$ bits for our ABA protocol; this violates out promised bound for ABA). Hence we present the code Output which is almost identical to protocol Output presented in section 14.2 and is just the customized version of protocol Output of Optimal-A-cast in the current settings.

Figure 14.8: Code for **Output Phase**

Output

i. CODE FOR P_i: Every party in $\mathcal{P}$ (not $\mathcal{P}'$) will execute this code.

1. If $P_i \in \mathcal{P}'_{ex}$, do the following to help the parties in $\overline{\mathcal{P}_{ex}} = \mathcal{P} \setminus \mathcal{P}'_{ex}$ to compute m^*_α:
   (a) Set $d = t' + 1$ and $c = \lceil \frac{\ell+1}{td} \rceil$.
   (b) Interpret m^*_α as a polynomial $p(x)$ of degree $d-1$ over $GF(2^c)$. For this, divide m^*_α into blocks of c bits and interpret each block as an element from $GF(2^c)$. These elements from $GF(2^c)$ are the coefficients of $p(x)$.
   (c) Send $p_i = p(i)$ to every $P_j \in \overline{\mathcal{P}_{ex}}$, where p_i is computed over $GF(2^c)$.
   (d) For every $P_j \in \overline{\mathcal{P}_{ex}}$, choose a random distinct hash key R_{ij} from $\mathbb{F}$ and send $(R_{ij}, \mathcal{X}_{ij1}, \ldots, \mathcal{X}_{ijn})$ to P_j, where for $k = 1, \ldots, n$, $\mathcal{X}_{ijk} = \mathcal{U}_\kappa(p_k, R_{ij})$. Here, to compute $\mathcal{X}_{ijk}$, interpret p_k as a c bit string.
   (e) Terminate this code with m^*_α as output.
2. If $P_i \in \overline{\mathcal{P}_{ex}}$, do the following to compute m^*_α:
   (a) Call p_k received from party $P_k \in \mathcal{P}'_{ex}$ as 'clean' if there are at least $t'+1$ P_j's in $\mathcal{P}'_{ex}$, corresponding to which $\mathcal{X}_{jik} = \mathcal{U}_\kappa(p_k, R_{ji})$ holds, where $(R_{ji}, \mathcal{X}_{ji1}, \ldots, \mathcal{X}_{jin})$ is received from $P_j \in \mathcal{P}'_{ex}$.
   (b) Wait to receive d 'clean' p_k's and upon receiving, interpolate $d-1$ degree polynomial $p(x)$ using those 'clean' values, interpret m^* from $p(x)$ and terminate this protocol with m^*_α as output.

Lemma 14.28 (Termination of the Output Phase) *An execution of* **Output Phase** *in any segment $\mathcal{S}_\alpha$ will terminate, except with error probability $\frac{\epsilon}{n}$.*

PROOF: Let E_{ex} and $E_{\overline{ex}}$ be the events that the honest parties in $\mathcal{P}'_{ex}$ and $\overline{\mathcal{P}_{ex}}$ (respectively) terminate. Here we show that $Prob(E_{ex}) = 1$ and $Prob(E_{\overline{ex}}) = (1 - \frac{\epsilon}{n})$. Consequently, we will have

$$\begin{aligned} Prob(\text{ Every honest party in } \mathcal{P} \text{ terminate}) &= Prob(E_{ex} \cap E_{\overline{ex}}) \\ &= Prob(E_{ex}) \cdot Prob(E_{\overline{ex}}) \\ &= 1 \cdot (1 - \frac{\epsilon}{n}) = (1 - \frac{\epsilon}{n}). \end{aligned}$$

From the steps of the code Output, the parties in $\mathcal{P}'_{ex}$ will always terminate after performing the steps as mentioned in step 1(a)-1(d) of the code. This asserts that $Prob(E_{ex}) = 1$.

So we now have to prove that $Prob(E_{\overline{ex}}) = (1-\frac{\epsilon}{n})$. To show this, we first assert that if all the honest parties in $\mathcal{P}'_{ex}$ holds common m^*_α, then $Prob(E_{\overline{\mathcal{P}'_{ex}}}) = 1$ holds; but the parties in $\mathcal{P}'_{ex}$ holds common m^*_α, except with probability $\frac{\epsilon}{n}$ (from Lemma 14.26). So, we have $Prob(E_{\overline{ex}}) = Prob(\text{honest parties in } \mathcal{P}'_{ex} \text{ hold common } m^*_\alpha) \cdot Prob(E_{\overline{ex}} \mid \text{honest parties in } \mathcal{P}'_{ex} \text{ hold common } m^*_\alpha) = (1 - \frac{\epsilon}{n}).1 = (1 - \frac{\epsilon}{n})$.

Now what is left is to show that $Prob(E_{\overline{ex}}) = 1$ when all the honest parties in $\mathcal{P}'_{ex}$ hold common m^*_α. Consider an honest party P_i in $\overline{\mathcal{P}_{ex}}$. Clearly, P_i will terminate if it receives $d = t' + 1$ 'clean' values eventually. To assert that P_i will indeed receive $d = t' + 1$ 'clean' values, we first show that the value p_k received from every honest P_k in $\mathcal{P}'_{ex}$ will be considered as 'clean' by P_i. Consequently, since there are $t' + 1$ honest parties in $\mathcal{P}'_{ex}$, P_i will eventually receive $t' + 1$ 'clean' values even though the corrupted parties in $\mathcal{P}'_{ex}$ never send any value to P_i. As the honest parties in $\mathcal{P}'_{ex}$ have common m^*_α, they will generate same $p(x)$ and therefore same $p_k = p(k)$. Hence, $\mathcal{X}_{jik} = \mathcal{U}_\kappa(p_k, R_{ji})$ will hold, with respect to $(R_{ji}, \mathcal{X}_{jik})$ of every honest P_j in $\mathcal{P}'_{ex}$. As there at least $d = t' + 1$ honest parties in $\mathcal{P}'_{ex}$, this proves that p_k received from honest $P_k \in \mathcal{P}'_{ex}$ will be considered as 'clean' by P_i. This proves $Prob(E_{\overline{ex}}) = 1$ when all the honest parties in $\mathcal{P}'_{ex}$ holds common m^*_α. Hence the lemma. □

Lemma 14.29 (Correctness of Output Phase) *Every honest party in $\mathcal{P}$ will output a common message m^*_α in an execution of* **Output Phase** *in a segment $\mathcal{S}_\alpha$, except with error probability $\frac{\delta}{n}$. Moreover, if the honest parties start $\mathcal{S}_\alpha$ with same input m_α, then $m^*_\alpha = m_\alpha$.*

Proof: Consider the following two events, E_{ex} and $E_{\overline{ex}}$, where E_{ex} is the event that *all the honest parties in $\mathcal{P}'_{ex}$ will output same m^*_α* and $E_{\overline{ex}}$ is the event that *all the honest parties in $\overline{\mathcal{P}_{ex}}$ will output same m^*_α*. We now show that $Prob(E_{ex}) = (1 - \frac{\delta}{n^2})$ and $Prob(E_{\overline{ex}} | E_{ex}) = (1 - \frac{\delta}{n})$.

1. $Prob(E_{ex}) = (1 - \frac{\delta}{n^2})$: Follows from Lemma 14.26.

2. $Prob(E_{\overline{ex}} | E_{ex}) = (1 - \frac{\delta}{n})$: Here we show that every honest $P_i \in \overline{\mathcal{P}_{ex}}$ will output m^*_α, except with probability $\frac{\delta}{n^2}$. This will assert that all the honest parties in $\overline{\mathcal{P}_{ex}}$ will output same m^*_α, except with error probability $|\overline{H}|\frac{\delta}{n^2}$ where $\overline{H}$ is the set of honest parties in $\overline{\mathcal{P}_{ex}}$. As $|\overline{H}|$ can be at most t, we have $|\overline{H}|\frac{\delta}{n^2} \approx \frac{\delta}{n}$.

   So let $P_i \in \overline{\mathcal{P}_{ex}}$ be an honest party. Now the p_k value of each honest $P_k \in \mathcal{P}'_{ex}$ will be eventually considered as 'clean' value by honest P_i. This is because there are at least $t' + 1$ honest parties in $\mathcal{P}'_{ex}$, who hold same m^*_α and therefore same $p(x)$ (and hence $p(k)$). So $\mathcal{X}_{jik} = \mathcal{U}_\kappa(p_k, R_{ji})$ will hold, with respect to $(R_{ji}, \mathcal{X}_{jik})$ of every honest P_j in $\mathcal{P}'_{ex}$. A corrupted $P_k \in \mathcal{P}'_{ex}$ may send $\overline{p_k} \neq p_k$ to P_i, but $\overline{p_k}$ will not be considered as a 'clean' value with probability at least $(1 - \frac{\delta}{n^3})$. This is because, in order to be considered as 'clean' value, $\overline{p_k}$ should satisfy $\mathcal{X}_{jik} = \mathcal{U}_\kappa(\overline{p_k}, R_{ji})$ with respect to $(R_{ji}, \mathcal{X}_{jik})$ of at least $t + 1$ P_j's from $\mathcal{P}'_{ex}$. The test will fail with respect to an honest party from $\mathcal{P}'_{ex}$ with probability $\frac{c2^{-\kappa}}{\kappa} \approx \frac{\delta}{n^4}$ according to **Collision Theorem** (replacing ℓ by c in Theorem 14.5, where $c = \lceil \frac{\ell+1}{td} \rceil = \lceil \frac{\ell+1}{tt'} \rceil$). Thus though the test may pass with respect to all corrupted parties in $\mathcal{P}'_{ex}$ (at most t), the test will fail for every honest party from $\mathcal{P}'_{ex}$ with probability $(1 - \frac{\delta}{n^4})^{|H|}$, where H is the set of honest parties in $\mathcal{P}'_{ex}$. Now since $|H|$ can be $\mathcal{O}(t)$, we have $(1 - \frac{\delta}{n^4})^{|H|} \approx (1 - |H|\frac{\delta}{n^4}) \approx (1 - \frac{\delta}{n^3})$. Now the probability that none of the wrong $\overline{p_k} \neq p_k$ sent by corrupted P_ks in $\mathcal{P}'_{ex}$ will be considered as 'clean' by honest P_i is $(1 - \frac{\delta}{n^3})^{\mathcal{O}(t)} \approx (1 - \mathcal{O}(t)\frac{\delta}{n^3}) \approx (1 - \frac{\delta}{n^2})$ (as there can be at most $\mathcal{O}(t)$ corrupted parties in $\mathcal{P}'_{ex}$). Hence, honest P_i will reconstruct $p(x)$ using d 'clean' values (which he is bound to get eventually), except with probability $\frac{\delta}{n^2}$.

So we have, $Prob($ Every honest party in $\mathcal{P}$ holds common $m^*_\alpha) = (1 - \frac{\delta}{n^2}) \cdot (1 - \frac{\delta}{n}) \approx (1 - \frac{\delta}{n})$. The second part is easy to follow. □

In the next section, we prove the properties of protocol Optimal-ABA.

14.3.3.4 Properties of Optimal-ABA

Lemma 14.30 *In Optimal-ABA, in total there can be t segment failures. The* **Checking Phase** *and* **Expansion Phase** *may be executed for at most $2t$ times. But* **Output Phase** *may be executed at most t times, once for each segment.*

Proof: As there are total t corrupted parties, in total there can be t segment failures. These t failures may occur with respect to a single segment or may be distributed across t segments. After t failures, all corrupted parties will be removed from $\mathcal{P}$ and hence segment failure can not occur any more.

Since a segment may fail in **Expansion Phase**, there can be $2t$ executions of **Checking Phase** and **Expansion Phase** of which at most t may be non-robust executions (conflict triplet is found) and remaining t may be robust executions. Since segment can not fail in **Output Phase**, this phase may be executed at most t times, once for each segment. □

Lemma 14.31 (Termination of Optimal-ABA) *Protocol Optimal-ABA will terminate eventually, except with error probability at most ϵ.*

Proof: Let T denote the event that Optimal-ABA terminates. Likewise T_α denotes the event that segment $\mathcal{S}_\alpha$ terminates. From protocol Optimal-ABA, we see that the segments are executed sequentially. That is, an honest party starts executing segment $S_{\alpha+1}$ only after it terminates segment $\mathcal{S}_\alpha$. We formalize this by introducing a dependency relation between any two events, say E_1 and E_2: We write $E_1 \mapsto E_2$ to mean that event E_2 occurs, given that event E_1 happens. Now we see that in protocol Optimal-ABA, the following holds: $T_1 \mapsto T_2 \mapsto \cdots \mapsto T_t$.

It is clear that protocol Optimal-ABA terminates implies that all the t segments $\mathcal{S}_1, \ldots, \mathcal{S}_t$ terminates and therefore we have

$$\begin{aligned}
Prob(T) &= Prob(T_t \cap T_{t-1} \cap \cdots \cap T_1) \\
&= Prob(T_t | (T_{t-1} \cap \cdots \cap T_1)) \cdot Prob(T_{t-1} \cap \cdots \cap T_1) \\
&\cdots\cdots\cdots \\
&= Prob(T_t | (T_{t-1} \cap \cdots \cap T_1)) \cdot Prob(T_{t-1} | (T_{t-2} \cap \cdots \cap T_1)) \cdots Prob(T_2 | T_1) \\
&\quad \cdot Prob(T_1)
\end{aligned}$$

So we now estimate each of the above probabilities in the last line of the above equation. Let a_α be the number of times **Checking Phase** and **Expansion Phase** has been executed in segment $\mathcal{S}_\alpha$. By Lemma 14.30, $t \leq a_1 + \ldots + a_t \leq 2t$. Recall that **Output Phase** will be executed only once in a segment. We now estimate $Prob(T_1)$. If segment $\mathcal{S}_1$ terminates then it implies that all the a_1 instances of **Checking Phase** and **Expansion Phase** and the single instance of **Output Phase** in $\mathcal{S}_1$ terminates. It also implies that event E (recall from subsection describing **Checking Phase**) happens in all the a_1 instances of **Checking Phase** in $\mathcal{S}_1$. We now define the following notations:

- E_i: denotes the event that event E happens in the i^{th} instance of **Checking Phase** in segment $\mathcal{S}_1$, for $i = 1, \ldots, a_1$.
- C_i: denotes the event that i^{th} instance of **Checking Phase** terminates in segment $\mathcal{S}_1$, for $i = 1, \ldots, a_1$.
- X_i: denotes the event that i^{th} instance of **Expansion Phase** terminates in segment $\mathcal{S}_1$, for $i = 1, \ldots, a_1$.
- O: denotes the event that the single instance of **Output Phase** terminates in $\mathcal{S}_1$.

In segment $\mathcal{S}_1$, the following dependency relation holds: $E_1 \mapsto C_1 \mapsto X_1 \mapsto E_2 \mapsto \cdots \mapsto E_{a_1} \mapsto C_{a_1} \mapsto X_{a_1} \mapsto O$. Now E_1 occurs with probability $(1 - \frac{\epsilon}{n})$. This follows from the fact that E occurs with probability $(1 - \frac{\epsilon}{n})$. Moreover any event in the above chain of events happens with probability $(1 - \frac{\epsilon}{n})$, given that all the events before that event has happened. This follows from Lemma 14.21, Lemma 14.25 and Lemma 14.28. Therefore, we may write

$$
\begin{aligned}
Prob(T_1) &= Prob(O \cap X_{a_1} \cap C_{a_1} \cap E_{a_1} \cap \cdots \cap X_1 \cap C_1 \cap E_1) \\
&= Prob(O|(X_{a_1} \cap C_{a_1} \cap E_{a_1} \cap \cdots \cap X_1 \cap C_1 \cap E_1) \cdot \\
&\quad\; Prob(X_{a_1} \cap C_{a_1} \cap E_{a_1} \cap \cdots \cap X_1 \cap C_1 \cap E_1) \\
&\quad\; \ldots\ldots\ldots \\
&\quad\; \ldots\ldots\ldots \\
&= Prob(O|(X_{a_1} \cap C_{a_1} \cap E_{a_1} \cap \cdots \cap X_1 \cap C_1 \cap E_1)) \cdot \cdots \\
&\quad\; Prob(X_1|(C_1 \cap E_1)) \cdot Prob(C_1|E_1) \cdot Prob(E_1) \\
&= (1 - \frac{\epsilon}{n}) \cdot (1 - \frac{\epsilon}{n}) \cdot \cdots\cdot \quad \text{— } (3a_1 + 1) \text{ times.} \\
&= (1 - \frac{\epsilon}{n})^{3a_1+1}
\end{aligned}
$$

Now it is easy to see that $Prob(T_2|T_1) = (1 - \frac{\epsilon}{n})^{3a_2+1}$ and in general $Prob(T_\alpha|(T_{\alpha-1} \cap T_{\alpha-2} \cap \cdots \cap T_1)) = (1 - \frac{\epsilon}{n})^{3a_\alpha+1}$. Hence we can write

$$
\begin{aligned}
Prob(T) &= Prob(T_t|(T_{t-1} \cap \cdots \cap T_1)) \cdot Prob(T_{t-1}|(T_{t-2} \cap \cdots \cap T_1)) \\
&\quad\; \cdots Prob(T_2|T_1) \cdot Prob(T_1) \\
&= (1 - \frac{\epsilon}{n})^{3a_t+1} \cdots (1 - \frac{\epsilon}{n})^{3a_1+1} \\
&= (1 - \frac{\epsilon}{n})^{\sum_{\alpha=1}^{t}(3a_\alpha+1)} \\
&= (1 - \frac{\epsilon}{n})^{7t} \quad \text{Putting } \sum_{\alpha=1}^{t} a_\alpha = 2t \\
&\approx (1 - 7t\frac{\epsilon}{n}) \\
&\approx (1 - \epsilon).
\end{aligned}
$$

Hence the lemma. □

Lemma 14.32 *Conditioned on the event that segment $\mathcal{S}_\alpha$ terminates, every honest party outputs common m^*_α at the end of $\mathcal{S}_\alpha$, except with probability $\frac{\delta}{n}$. Moreover if the honest parties start $\mathcal{S}_\alpha$ with same input message m_α, then $m^*_\alpha = m_\alpha$.*

PROOF: $\mathcal{S}_\alpha$ may terminate at the end of **Checking Phase** or at the end of **Output Phase**. If $\mathcal{S}_\alpha$ terminates at the end of **Checking Phase**, then every party assigns $m^*_\alpha = m^\dagger_\alpha$, where $m^\dagger_\alpha$ is a predefined value. Hence in this case the first part of the lemma holds without any error.

Now let $\mathcal{S}_\alpha$ terminates at the end of **Output Phase**. Let us define an event E_o: E_o is the event that every party in $\mathcal{P}$ outputs common m^*_α at the end of **Output Phase** of $\mathcal{S}_\alpha$. Now by **Correctness of the Output Phase** (Lemma 14.29), given event E, all honest parties in $\mathcal{P}$ will hold common m^*_α, except with error probability $\frac{\delta}{n}$. So we have, $Prob(E_o|E) \geq (1 - \frac{\delta}{n})$. Now

$$\begin{aligned} Prob(E_o) &= Prob(E_o|E) \cdot Prob(E) \\ &\geq (1 - \frac{\delta}{n})(1 - \frac{\epsilon}{n}) \geq 1 - \frac{\delta}{n} - \frac{\epsilon}{n} \\ &\geq 1 - \frac{\delta}{n} - \frac{\delta}{n^2} \text{ as } \epsilon \leq \frac{\delta}{n} \\ &\approx 1 - \frac{\delta}{n} \end{aligned}$$

We now prove the second part of the lemma. If all the honest parties start with same input m_α then by Lemma 14.22, $\mathcal{P}'_{ch}$ will be constructed with the honest parties in it holding m_α. Then by Lemma 14.26, $\mathcal{P}'_{ex}$ will be constructed with m_α as the common message of the honest parties in it and finally every honest party in $\mathcal{P}$ will agree on m_α in **Output Phase**. □

Lemma 14.33 (Correctness of Optimal-ABA) *Conditioned on the event that Optimal-ABA terminates, every honest party outputs common m^* at the end of Optimal-ABA, except with probability δ. Moreover if the honest parties start Optimal-ABA with same input message m, then $m^* = m$.*

PROOF: By Lemma 14.32, every honest party outputs common m^*_α at the end of $\mathcal{S}_\alpha$, except with probability $\frac{\delta}{n}$, conditioned on the event that $\mathcal{S}_\alpha$ terminates. As there are t segments, for all the segments the above holds, except with probability at most $t\frac{\delta}{n} \approx \delta$. Since m^* is the concatenation of $m^*_1, \ldots, m^*_t$, it follows that every party agrees on common m^*, except with probability δ.

The second part of the lemma follows from the fact that if the honest parties start $\mathcal{S}_\alpha$ with same input message m_α, then $m^*_\alpha = m_\alpha$ (by Lemma 14.32). □

Theorem 14.34 *Optimal-ABA is a (ϵ, δ)-ABA protocol.*

PROOF: Follows from Lemma 14.31 and Lemma 14.33. □

Theorem 14.35 *Protocol Optimal-ABA privately communicates $\mathcal{O}(\ell n + n^{10}\kappa + n^7\kappa^2)$ bits to agree on an ℓ bit message.*

PROOF: In Optimal-ABA, **Checking Phase** and **Expansion Phase** may be executed for at most $2t$ times and **Output Phase** may be executed t times (by Lemma 14.30). We now compute the communication complexity of a *single* execution of **Checking Phase** and **Expansion Phase**. In **Checking Phase**, there are at most $2n'^2$ instances of SAVSS-Share and SAVSS-Rec-Private. Moreover, there are two executions of Patra-ACS to agree on a set of parties of size $t+1$ and n' A-cast of n length response vectors. Since $n' = \mathcal{O}(n)$, the total communication complexity during one execution of **Checking Phase** is $\mathcal{O}(n^8\kappa+n^6(\kappa^2+n\log n))$ bits.

During the execution of **Expansion Phase**, the most costly step in terms of communication complexity is the execution of Patra-ACS, which will be executed t' times (the maximum number of iterations of `while` loop) in the `while` loop. Since $t' = \mathcal{O}(n)$, this step requires a communication complexity of $\mathcal{O}(n^9\kappa)$ bits. Moreover, during **Expansion Phase** each party in $\mathcal{K}$ will privately send his ℓ/t bit message to exactly one party in $\overline{\mathcal{K}}$ to which it is mapped. As $|\mathcal{K}| = \mathcal{O}(n)$, this step requires a communication cost of $\mathcal{O}(n\ell/t)$ bits. So in total, the communication complexity of a *single* execution of **Checking Phase** plus **Expansion Phase** is $\mathcal{O}(n^9\kappa + n^6\kappa^2 + n\ell/t)$ bits. So executing both the phases $2t = \Theta(n)$ times require a communication complexity of $\mathcal{O}(\ell n+n^{10}\kappa+n^7\kappa^2)$ bits.

A single execution of **Output Phase** requires $\mathcal{O}(n'^2c + n'^3\kappa)$ bits of private communication. Now $\mathcal{O}(n'^2c+n'^3\kappa) = \mathcal{O}(\ell+n'^3\kappa)$ as $c = \lceil\frac{\ell+1}{td}\rceil = \lceil\frac{\ell+1}{tt'}\rceil$ and $n' = \mathcal{O}(n)$, $t' = \mathcal{O}(n)$. So t executions of **Output Phase** require a communication complexity of $\mathcal{O}(n\ell+n^4\kappa)$ bits. Thus the communication complexity of Optimal-ABA is $\mathcal{O}(\ell n + n^{10}\kappa + n^7\kappa^2)$ bits. □

14.4 Conclusion and Open Problem

In this chapter, we presented communication optimal multi-valued A-cast and ABA protocols for large message. Specifically, our protocols for A-cast and ABA are *communication optimal* for any $\ell = \omega(n^2(n+\kappa))$ and $\ell = \omega(n^9\kappa + n^6\kappa^2)$, respectively. As far as our knowledge is concerned, we are the first to propose communication optimal protocols for large message in asynchronous networks. Our ABA protocol uses party-elimination framework and introduces a novel technique to construct a set containing $2t+1$ parties with all honest parties in it holding a common message m, from a set of $t+1$ parties with all the honest party(ies) in it holding m.

As mentioned before in this chapter, we can get a better bound on ℓ (for which our ABA protocol is communication optimal) by replacing the black box ABA of [127] by more communication efficient ABA protocol for small message. But designing such efficient ABA protocol (which is also an important problem in its own right) is beyond the scope of this chapter and we leave it as an interesting open question. A slightly tougher problem would be to design communication optimal A-cast and ABA protocols for all values of ℓ (if it is possible to design). In summary, we have the following open question:

Open Problem 22 *How to design communication optimal A-cast and ABA protocols for all values of ℓ? or We may ask: Is it possible to design communication optimal A-cast and ABA protocols for all values of ℓ?*

Part III

Summary, Discussions and Future Directions

Chapter 15

Conclusion

In this chapter, we summarize our contribution in this thesis, draw several insightful inferences from our investigations and then mention several problems for future investigations.

15.1 Summary of Contributions

In this thesis, three related secure distributed computing problems were studied: VSS, BA and MPC. We first briefly summarize our main achievements concerning VSS:

- We have investigated the round complexity of statistical VSS in synchronous network and have shown that the lower bounds for the round complexity of perfect VSS can be circumvented by introducing a negligible probability of error. We also have shown that the above can be concluded even for the weaker notion of VSS, called WSS which is used as important building block for VSS.
- We have designed statistical VSS protocol with optimal resilience in synchronous network which reports the best known communication complexity and round complexity so far in the literature.
- We have proposed several AVSS protocols both in perfect and statistical category which show significant gain in communication complexity over the existing protocols.

For designing our VSS protocols in synchronous and asynchronous network, we have also investigated the complexity measures of ICP and have proposed several protocols in both the networks. Our protocols are much more efficient than the existing ICPs. An important feature of our VSS and ICP protocols are that all of them can deal with multiple secrets *concurrently* (if necessary) and thus harness many advantages of dealing with multiple secrets. Though here we have used our ICPs and VSS protocols to design ABA, MPC and AMPC protocols, they are of independent interest and they can be used in many other applications.

We next summarize our contribution for BA. Our works in this thesis focus on asynchronous BA or ABA.

- We have proposed a new *optimally resilient* ABA protocol for short message that improves the communication complexity of existing ABA protocols significantly. In that direction, we have twisted the existing common coin protocol that can now use an AVSS protocol sharing multiple secrets *concurrently* (the existing common coin protocol was capable of using an AVSS for single secret only).

- We have also proposed *communication optimal and optimally resilient* A-cast and ABA protocols for sufficiently long message (i.e both the protocols communicates $\mathcal{O}(\ell n)$ bits for a message of size ℓ).

Finally, the main achievements for MPC are as follows:

- Communication and round complexity being two most important parameters of MPC protocols, we have presented a statistical MPC protocol in synchronous network that simultaneously minimizes the communication and round complexity, while the existing protocols either focuses on reducing communication complexity or round complexity at a time.
- We have studied a specific instance of MPC problem (in synchronous network) called multiparty set intersection or MPSI problem. Here we have shown the drawbacks of the existing MPSI protocol and proposed two new protocols for the problem.
- In asynchronous network, we have proposed both statistical as well as perfect AMPC protocols. Our protocol for statistical AMPC with optimal resilience shows a huge gain in communication complexity over the existing protocol. Even our perfect AMPC with optimal resilience gains over the existing protocol in terms of communication complexity. Apart from presenting protocols, we have also shown that the existing AMPC that reported to have the best communication complexity (same as ours) is not a correct AMPC protocol.

15.2 Insightful Inferences

- Our investigation on the round complexity of statistical VSS and WSS in synchronous network has shown that the lower bounds for the round complexity of VSS and WSS can be circumvented by introducing a negligible probability of error. For instance, with $3t+1$ parties while perfect VSS requires *three* rounds for sharing, statistical VSS requires only *two* rounds.
- In general, designing any interactive protocol in asynchronous network is much more difficult and un-intuitive than designing protocol in synchronous network. Also due to inherent difficulties, most of the interactive distributed computing protocols are generally less fault tolerant and more communication intensive in asynchronous network than synchronous network. For example, while perfect AVSS and AMPC is possible iff $n \geq 4t+1$, perfect VSS and MPC is possible iff $n \geq 3t+1$. But unlike AVSS and AMPC, both BA as well as ABA require $n \geq 3t+1$. Now it is natural to expect a gap in communication complexity (or communication optimality) between BA and ABA protocols with $n = 3t+1$. But our result in this work shows that asynchrony of the network seems to have no effect on communication optimality as well. Fitzi et al. [75] designed a *communication optimal* (i.e

communicates $\mathcal{O}(\ell n)$ bits for a message of size ℓ) BA for large message with the help of BA protocols for smaller message in *synchronous network*. In this work, we have achieved the same for ABA. Our ABA protocol for long message extracts several advantages offered by directly dealing with long messages. In our ABA, we use *player-elimination* framework introduced in [98] in the context of MPC. So far player-elimination was used only in MPC and AMPC and hence our result shows the first non-MPC application of the technique.

- As mentioned in the previous item, perfect AVSS and AMPC is possible iff $n \geq 4t+1$ parties, while perfect VSS and MPC is possible iff $n \geq 3t+1$. So perfect AVSS and AMPC are less fault tolerant in comparison to perfect VSS and MPC. Thus we can at least work towards closing the gap in communication complexity of the protocols in synchronous and asynchronous network. In our work, we are almost able to close the gap between perfect MPC and AMPC with optimal resilience. While the best known perfect MPC [14] requires communication of $\mathcal{O}(n \log n)$ bits per multiplication gate, our proposed perfect AMPC requires $\mathcal{O}(n^2 \log n)$ bits per multiplication gate (existing best known AMPC reported $\mathcal{O}(n^3 \log n)$). So this urges one to further investigate for closing the gap completely. For designing our AMPC, we present a novel AVSS scheme that achieves an interesting property which is first of its kind.

- Similar to perfect case mentioned before, statistical AVSS and AMPC are less fault tolerant than statistical VSS and MPC. But unlike perfect case, there was a huge gap in communication complexity between statistical MPC and AMPC with optimal resilience (statistical MPC: $\mathcal{O}(n^2 \log \frac{1}{\epsilon})$ bits per multiplication gate [12]; statistical AMPC: $\Omega(n^{11}(\log \frac{1}{\epsilon})^4)$ bits per multiplication gate [21]). In this work, we have significantly reduced this gap by presenting a statistical AMPC that communicates $\mathcal{O}(n^5 \log \frac{1}{\epsilon})$ bits per multiplication gate. But compared to perfect case, statistical case requires more effort in order to close the gap. The key primitive of our AMPC is ACSS which is designed using AVSS as the vital component. The only existing optimally resilient statistical AVSS of [39] uses many black box primitives for its design and due to the use of many primitives, the protocol is very involved and communication intensive. For our statistical AVSS, we not only reduced the number of used black-box primitives, but we also provide efficient implementation of our primitives. Put together, our proposed statistical AVSS protocol is far better than the existing protocol of [39].

- We have studied a specific instance of MPC problem called multiparty set intersection or MPSI problem and provided customized solutions for the same. From our investigation, we conclude that though MPSI can be solved using general MPC protocol, but a general MPC may not give as efficient solution as a specific solution to these problems may provide. This is because, the specific solution takes into account subtleties of the problems and accordingly finds efficient solution. Hence the well-motivated specific instances of MPC, e.g. set union, cardinality, matching etc, may be looked at individually in order to obtain customized solutions.

- Our proposed optimally resilience statistical VSS (in synchronous network) reporting the best known communication and round complexity employs an important primitive called ICP. In this work, we have extended the existing definition of ICP to a multi-verifier and multi-secret ICP and proposed a protocol for the same which turns out to be the best one in the literature in terms of both communication and round complexity. Using our new multi-verifier and multi-secret ICP (along with several other new techniques), our VSS is able share multiple secrets concurrently and is far better that multiple execution of protocols for single secret. Finally using our VSS for multiple secrets we are able to design a synchronous MPC that minimizes the communication and round complexity simultaneously, where existing MPC protocols try to minimize one complexity measure at a time.

15.3 Future Works and Future Directions

Several open problems have been mentioned in-text throughout the thesis. However, all these are what are not yet solved within the purview and viewpoint of this thesis. In this section, we provide some future directions that are beyond this work's viewpoint. We note that there are several ways to look beyond this work and they can be categorized as

1. **Future Work of Type I:** Future work that increases the scope of the results and that weakens the set of assumptions made in this thesis.
2. **Future Work of Type II:** Future work that incorporates several concepts from across the fields like game theory to redefine the concept of our concerned problem.

We briefly discuss about the above items in the sequel.

15.3.1 Future Work of Type I

15.3.1.1 Incomplete Network

In this thesis, we restricted our network to complete network. But in real life it is quite impractical to have an underlying network as complete graph where every two parties can communicate between them though their private and secure channel. There have been very little attempt to characterize, find feasibility conditions and feasible solutions for the distributed computing problems like VSS, BA and MPC over graphs of smaller degree [55, 138, 16, 17, 88]. Thus it is very important to pursue further research in incomplete network model.

15.3.1.2 Directed Network

We have restricted our underlying network to be undirected which means every channel in the network provides bidirectional communication. Now if the network is incomplete as well as the channels are unidirectional, then we may ask the following questions: what are the conditions required for VSS, BA and MPC to be possible or feasible? The problem of secure and reliable message transmission (where there are specific processors called sender and receiver in distributed network such that the sender wants to send some message securely and reliably to the receiver) have been studied in incomplete directed network in [144, 142]. Directed network is essential to model several practical situations. Hence it is important to study VSS, BA and MPC in directed network.

15.3.1.3 Mobile Network

While the topology of the network is assumed to remain unchanged throughout the runtime of the protocol in this thesis, such an assumption may not always hold in practice. Thus, secure computation over mobile networks seems to be a different proposition all together. We note that the ideas to deal with mobile adversaries over static networks can be used to thwart static (or mobile) adversaries in mobile networks (over the same set of parties) - this is because, the disappearing of an edge can be treated as that edge being newly fail-stop corrupted, while the reappearing of a new edge may be treated as the curing of that infected edge.

15.3.1.4 Hybrid Network

In this thesis we have considered purely synchronous or purely asynchronous network. There can be networks that exercise properties of synchronous and asynchronous network in many ways. In literature there are works on MPC, VSS and BA over network that has several synchronous rounds in the beginning and subsequently the network behaves in a complete asynchronous manner [15, 79]. There is another work on MPC where the network is assumed to have a synchronization point, the network behaves as asynchronous network before and after the point [51]. We may define practically motivated hybrid networks of different types and study the problems in such networks.

15.3.1.5 Distributed Topology Information

In most situations we assume that every party is well aware of the topology of the network used for communication (applicable for incomplete network). But unlike the full-topological-knowledge model, a more realistic model is the one wherein each party knows about all his neighbors in the network. Or sometimes, each party knows up to two levels that is his neighbors' neighbors, or may be up to some constant number of levels. We remark that the design of secure protocols with distributed topological information (like each party knowing the identity of his neighbors alone) is several times more challenging and complex as compared to their full-topological-information counterparts.

15.3.1.6 Fixed Fault Quality and Quantity

Not all basic fault-types are covered by our fault model and the models used so far in the literature of VSS, BA, MPC. We have considered Byzantine adversary only for this thesis. There are many works that deal with passive, mobile or mixed adversary [20, 124, 33, 77, 99, 76, 2]. But there are other fault types that are never explored in the literature. For example, the disruptive fault wherein data integrity is lost without the adversary actually reading the data is not considered in this thesis and even in the literature of VSS, BA, MPC.

15.3.2 Future Work of Type II

15.3.2.1 Rational Distributed Computing

Motivated by the desire to develop more realistic models of, and protocols for, interactions between mutually distrusting parties, there has recently been significant interest in combining the approaches and techniques of game theory with those of cryptographic protocol design. This inter-disciplinary area of research is also called as Rational Cryptography. So we may further study rational distributed computing that would include rational VSS, BA and MPC. Broadly speaking, in rational cryptography two directions are currently being pursued in this area (we may recast the following in terms of VSS, BA and MPC problem):

1. Applying cryptography to game theory: Certain game-theoretic equilibria are achievable if a trusted mediator is available. The question here is: to what extent can this mediator be replaced by a distributed cryptographic protocol run by the parties themselves?

2. Applying game-theory to cryptography: Traditional cryptographic models assume some honest parties who faithfully follow the protocol, and some arbitrarily malicious parties against whom the honest parties must be protected. Game-theoretic models propose instead that all parties are simply self-interested (i.e., rational), and the question then is: how can we model and design meaningful protocols for such a setting?

15.3.2.2 Quantum Distributed Computing

Quantum cryptography is different from traditional cryptographic systems in that it relies more on physics, rather than mathematics, as a key aspect of its security model. An important and unique property of quantum cryptography is the ability of the two communicating users to detect the presence of any third party trying to gain knowledge of the key. This results from a fundamental aspect of quantum mechanics: the process of measuring a quantum system in general disturbs the system. A third party trying to eavesdrop on the key must in some way measure it, thus introducing detectable anomalies. By using quantum superpositions or quantum entanglement and transmitting information in quantum states, a communication system can be implemented which detects eavesdropping. If the level of eavesdropping is below a certain threshold, a key can be produced that is guaranteed to be secure (i.e. the eavesdropper has no information about), otherwise no secure key is possible and communication is aborted. The security of quantum cryptography relies on the foundations of quantum mechanics, in contrast to traditional public key cryptography which relies on the computational difficulty of certain mathematical functions, and cannot provide any indication of eavesdropping or guarantee of key security. There are already huge amount of work on VSS, BA and MPC in quantum cryptography settings. We may further study these problems.

List of Publications Related to This Thesis (In reverse chronological order)

1. **Arpita Patra**, Ashish Choudhary, and C. Pandu Rangan. Communication Efficient Perfectly Secure VSS and MPC in Asynchronous Networks with Optimal Resilience. In D. J. Bernstein and T. Lange, editors, *Advances in Cryptology - AFRICACRYPT'10, Third International Conference in Cryptology in Africa, Stellenbosch, South Africa, May 3-6, 2009, Proceedings*, volume 6055 of *Lecture Notes in Computer Science*, pages 184–202. Springer Verlag, 2010. Full version communicated to *Journal of ACM (JACM).*

2. **Arpita Patra**, Ashish Choudhary, and C. Pandu Rangan. Efficient Statistical Asynchronous Verifiable Secret Sharing with Optimal Resilience. In K. Kurosawa, editor, *Fourth International Conference on Information Theoretic Security, ICITS 2009, Shizuoka, Japan, December 3-6, 2009, Proceedings*, volume 5973 of *Lecture Notes in Computer Science*, Springer Verlag, 2009. Full version communicated to Journal of *Information and Computation.*

3. **Arpita Patra**, Ashish Choudhary and C. Pandu Rangan. Communication Efficient Statistical Asynchronous Multiparty Computation with Optimal Resilience. Accepted in *The 5th International Conferences on Information Security and Cryptology (INSCRYPT) 2009, December 12-15, 2009, Beijing, China, Lecture Notes in Computer Science.*

4. **Arpita Patra**, Ashish Choudhury, Tal Rabin, and C. Pandu Rangan. The Round Complexity of Verifiable Secret Sharing Revisited. In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 487–504. Springer Verlag, 2009.

5. **Arpita Patra**, Ashish Choudhary and C. Pandu Rangan. Efficient Asynchronous Byzantine Agreement with Optimal Resilience. In L. Alvisi, editor, *Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing, PODC 2009, Calgary, Alberta, Canada, August 10-12,* pages 92-101, ACM Press, 2009. Full version communicated to Journal of *Distributed Computing.*

6. **Arpita Patra**, Ashish Choudhary and C. Pandu Rangan. Round Efficient Unconditionally Secure MPC and Multiparty Set Intersection with Optimal Resilience. In *Progress in Cryptology - INDOCRYPT 2009, 10th International Conference on Cryptology in India, New Delhi, India, December 13-16, 2009. Proceedings*, volume 5922 of *Lecture Notes in Computer Science*, pages 398–417. Springer Verlog, 2009. Full version communicated to Journal of *Theoretical Computer Science.*

7. **Arpita Patra**, Ashish Choudhary and C. Pandu Rangan. Information Theoretically Secure Multi Party Set Intersection Re-visited. In M. J. Jacobson Jr., V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Proceedings*, volume 5867 of *Lecture*

Notes in Computer Science. Springer Verlag, 2009. Full version communicated to *Journal of Designs, Codes and Cryptography.*

8. G. Sathya Narayanan, T. Aishwarya, Anugrah Agrawal, **Arpita Patra**, Ashish Choudhary and C. Pandu Rangan. Multi Party Distributed Private Matching, Set Disjointness and Cardinality Set Intersection with Information Theoretic Security. In J. A. Garay, A. Miyaji and A. Otsuka, editors, *In Proc. of 8th International Conference on Cryptology and Network Security (CANS) 2009, Kanazawa, Japan, December 12-14, 2009. Proceedings* volume 5888 of *Lecture Notes in Computer Science*, pages 21-40, Springer Verlag, 2009.

9. **Arpita Patra**, Ashish Choudhary and C. Pandu Rangan. Round efficient unconditionally secure multiparty computation protocol. In D. R. Chowdhury, V. Rijmen, and A. Das, editors, *Progress in Cryptology - INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings*, volume 5365 of *Lecture Notes in Computer Science*, pages 185–199. Springer Verlag, 2008.

10. **Arpita Patra** and C. Pandu Rangan. Communication Optimal Multi-Valued Asynchronous Byzantine Agreement with Optimal Resilience. Cryptology ePrint Archive, Report 2009/433, 2009.

11. **Arpita Patra** and C. Pandu Rangan. Communication Optimal Multi-Valued Asynchronous Broadcast Primitive with Optimal Resilience. Submitted to Latincrypt, 2010.

12. **Arpita Patra** and C. Pandu Rangan. Communication Efficient Asynchronous Byzantine Agreement. Submitted to PODC 2010.

13. **Arpita Patra** and C. Pandu Rangan. Communication and Round Efficient Information Checking Protocol. Communicated to *Information Processing Letter Journal.*

14. Jonathan Katz, Ranjit Kumaresan, Ashish Choudhary, Srivatsan Narayanan, **Arpita Patra**, Ananth Raghunathan, C. Pandu Rangan. The Round Complexity of Verifiable Secret Sharing: The Statistical Case. To be submitted.

Bibliography

[1] I. Abraham, D. Dolev, and J. Y. Halpern. An Almost-surely Terminating Polynomial Protocol for Asynchronous Byzantine Agreement with Optimal Resilience. In R. A. Bazzi and B. Patt-Shamir, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*, pages 405–414. ACM Press, 2008.

[2] B. Altmann, M. Fitzi, and U. M. Maurer. Byzantine Agreement Secure against General Adversaries in the Dual Failure Model. In P. Jayanti, editor, *Distributed Computing, 13th International Symposium, Bratislava, Slavak Republic, September 27-29, 1999, Proceedings*, volume 1693 of *Lecture Notes in Computer Science*, pages 123–137. Springer Verlag, 1999.

[3] J. Bar-Ilan and D. Beaver. Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, August 14-16, 1989, Edmonton, Alberta, Canada*, pages 201–209. ACM Press, 1989.

[4] D. Beaver. Multiparty Protocols Tolerating Half Faulty Processors. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 560–572. Springer Verlag, 1989.

[5] D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer Verlag, 1991.

[6] D. Beaver. Secure Multiparty Protocols and Zero-knowledge Proof Systems Tolerating a Faulty Minority. *Journal of Cryptology*, 4(4):75–122, 1991.

[7] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Security with Low Communication Overhead. In A. Menezes and S. A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 62–76. Springer Verlag, 1990.

[8] D. Beaver and S. Haber. Cryptographic Protocols Provably Secure Against Dynamic Adversaries. In R. A. Rueppel, editor, *Advances in Cryptology*

- *EUROCRYPT '92, Workshop on the Theory and Application of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, volume 658 of *Lecture Notes in Computer Science*, pages 307–323. Springer Verlag, 1992.

[9] D. Beaver, S. Micali, and P. Rogaway. The Round Complexity of Secure Protocols (Extended Abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513. ACM Press, 1990.

[10] D. Beaver and A. Wool. Quorum-Based Secure Multi-party Computation. In K. Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998. Proceeding,*, volume 1403 of *Lecture Notes in Computer Science*, pages 375–390. Springer Verlog, 1998.

[11] Z. Beerliová-Trubíniová, M. Fitzi, M. Hirt, U. M. Maurer, and V. Zikas. MPC vs. SFE: Perfect Security in a Unified Corruption Model. In R. Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings*, volume 4948 of *Lecture Notes in Computer Science*, pages 231–250. Springer Verlag, 2008.

[12] Z. Beerliová-Trubíniová and M. Hirt. Efficient Multi-party Computation with Dispute Control. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 305–328. Springer Verlag, 2006.

[13] Z. Beerliová-Trubíniová and M. Hirt. Simple and Efficient Perfectly-Secure Asynchronous MPC. In K. Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 376–392. Springer Verlag, 2007.

[14] Z. Beerliová-Trubíniová and M. Hirt. Perfectly-Secure MPC with Linear Communication Complexity. In R. Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer Verlag, 2008.

[15] Z. Beerliova-Trubiniova, M. Hirt, and J. B. Nielsen. Almost-Asynchronous MPC with Faulty Minority. Cryptology ePrint Archive, Report 2008/416, 2008.

[16] A. Beimel. On Private Computation in Incomplete Networks. In A. Pelc and M. Raynal, editors, *Structural Information and Communication Complexity, 12th International Colloquium, SIROCCO 2005, Mont Saint-Michel, France, May 24-26, 2005, Proceedings*, volume 3499 of *Lecture Notes in Computer Science*, pages 18–33. Springer Verlog, 2005.

[17] Amos Beimel. On Private Computation in Incomplete Networks. *Distributed Computing*, 19(3):237–252, 2007.

[18] M. Ben-Or. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. In *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Princiles of Distributed Computing, August 17-19, 1983, Montreal, Quebec, Canada*, pages 27–30. ACM Press, 1983.

[19] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous Secure Computation. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, 1993*, pages 52–61. ACM Press, 1993.

[20] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM Press, 1988.

[21] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous Secure Computations with Optimal Resilience. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17*, pages 183–192. ACM Press, 1994.

[22] J. C. Benaloh and J. Leichter. Generalized Secret Sharing and Monotone Functions. In S. Goldwasser, editor, *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 27–35. Springer Verlog, 1988.

[23] P. Berman, G. A. Garay, and K. J. Perry. Bit Optimal Distributed Consensus. In Computer Science Research, 2009.

[24] P. Berman and J. A. Garay. Asymptotically Optimal Distributed Consensus (Extended Abstract). In G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium, ICALP'89, Stresa, Italy, July 11-15, 1989, Proceedings*, volume 372 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 1989.

[25] P. Berman and J. A. Garay. Cloture Votes: n/4-Resilient Distributed Consensus in t+1 Rounds. *Mathematical Systems Theory*, 26(1):3–19, 1993.

[26] P. Berman, J. A. Garay, and K. J. Perry. Towards Optimal Distributed Consensus (Extended Abstract). In *Proceedings of 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, 30 October - 1 November 1989*, pages 410–415. IEEE Computer Society, 1989.

[27] G. R. Blakley. Safeguarding Cryptographic Keys. In *Proceedings of the National Computer Conference*, pages 313–317. American Federation of Information Processing Societies, 1979.

[28] M. Blum, A. D. Santis, S. Micali, and G. Persiano. Noninteractive Zero-Knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

[29] G. Bracha. An Asynchronous $\lfloor (n-1)/3 \rfloor$-resilient Consensus Protocol. In *Proceedings of the Third Annual ACM Symposium on Princiles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 154 – 162. ACM Press, 1984.

[30] G. Bracha. Asynchronous Byzantine Agreement Protocols. *Information and Computation*, 75(2):130–143, 1987.

[31] G. Bracha. An $O(\log n)$ Expected Rounds Randomized Byzantine Generals Protocol. *Journal of ACM*, 34(4):910–920, 1987.

[32] G. Bracha and S. Toueg. Asynchronous Consensus and Broadcast Protocols. *Journal of ACM*, 32(4):824–840, 1985.

[33] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-Round Secure Computation and Secure Autonomous Mobile Agents. In U. Montanari, José D. P. Rolim, and E. Welzl, editors, *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, volume 1853 of *Lecture Notes in Computer Science*, pages 512–523. Springer Verlog, 2000.

[34] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems. In V. Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washingtion, DC, USA, November 18-22, 2002*, pages 88–97. ACM Press, 2002.

[35] R. Canetti. *Studies in Secure Multiparty Computation and Applications.* PhD thesis, Weizmann Institute, Israel, 1995.

[36] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[37] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively Secure Multi-Party Computation. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 639–648. ACM Press, 1996.

[38] R. Canetti and A. Herzberg. Maintaining Security in the Presence of Transient Faults. In Y. Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 425–438. Springer Verlog, 1994.

[39] R. Canetti and T. Rabin. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 42–51. ACM Press, 1993.

[40] L. Carter and M. N. Wegman. Universal Classes of Hash Functions. *Journal of Computer and System Sciences*, 18(4):143–154, 1979.

[41] D. Chaum, C. Crpeau, and I. Damgård. Multiparty Unconditionally Secure Protocols (Extended Abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA. ACM 1988*, pages 11–19. ACM Press, 1988.

[42] D. Chaum, I. Damgård, and J. van de Graaf. Multiparty Computations Ensuring Privacy of Each Party's Input and Correctness of the Rsesult. In C. Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 87–119. Springer Verlag, 1987.

[43] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 383–395. ACM Press, 1985.

[44] B. A. Coan and J. L. Welch. Modular Construction of a Byzantine Agreement protocol with Optimal Message Bit Complexity. *Information and Computation*, 97(1):61–85, 1992.

[45] J. Considine, M. Fitzi, M. K. Franklin, L. A. Levin, U. M. Maurer, and D. Metcalf. Byzantine Agreement Given Partial Broadcast. *Journal of Cryptology*, 18(3):191–217, 2005.

[46] R. Cramer and I. Damgård. *Multiparty Computation, An Introduction*. Contemporary Cryptography. Birkhuser Basel, 2005.

[47] R. Cramer, I. Damgård, and S. Dziembowski. On the Complexity of Verifiable Secret Sharing and Multiparty Computation. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA. ACM*, pages 325–334. ACM Press, 2000.

[48] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient Multiparty Computations Secure Against an Adaptive Adversary. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326. Springer Verlag, 1999.

[49] R. Cramer, I. Damgård, and S. Fehr. On the Cost of Reconstructing a Secret, or VSS with Optimal Reconstruction Phase. In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 503–523. Springer Verlag, 2001.

[50] R. Cramer, I. Damgård, and U. M. Maurer. General Secure Multi-party Computation from any Linear Secret Sharing Scheme. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer Verlag, 2000.

[51] I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen. Asynchronous Multiparty Computation: Theory and Implementation. In S. Jarecki and G. Tsudik, editors, *Proceedings of Public Key Cryptography - PKC 2009,*

12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, volume 5443 of *Lecture Notes in Computer Science*, pages 160–179. Springer Verlag, March 2009.

[52] I. Damgård and J. B. Nielsen. Scalable and Unconditionally Secure Multiparty Computation. In A. Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer Verlag, 2007.

[53] Y. Desmedt and Y. Frankel. Threshold Cryptosystems. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer Verlag, 1989.

[54] D. Dolev. The Byzantine Generals Strike Again. *Journal of Algorithms*, 3(1):14–30, 1982.

[55] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly Secure Message Transmission. *Journal of ACM*, 40(1):17–47, 1993.

[56] D. Dolev, M. J. Fischer, R. J. Fowler, N. A. Lynch, and H. R. Strong. An Efficient Algorithm for Byzantine Agreement without Authentication. *Information and Control*, 52(3):257–274, 1982.

[57] D. Dolev and R. Reischuk. Bounds on Information Exchange for Byzantine Agreement. *Journal of ACM*, 32(1):191–204, 1985.

[58] D. Dolev, R. Reischuk, and H. R. Strong. 'Eventual' Is Earlier than 'Immediate'. In *Proc. of 23rd Annual Symposium on Foundations of Computer Science, 3-5 November 1982, Chicago, Illinois, USA*, pages 196–203. IEEE Press, 1982.

[59] D. Dolev, R. Reischuk, and H. R. Strong. Early Stopping in Byzantine Agreement. *Journal of ACM*, 37(4):720–741, 1990.

[60] D. Dolev and H. R. Strong. Polynomial Algorithms for Multiple Processor Agreement. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 401–407. ACM Press, 1982.

[61] D. Dolev and H. R. Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM Journal of Computing*, 12(4):656–666, 1983.

[62] C. Dwork. Strong Verifiable Secret Sharing (Extended Abstract). In J. van Leeuwen and N. Santoro, editors, *Distributed Algorithms, 4th International Workshop, WDAG '90, Bari, Italy, September 24-26, 1990, Proceedings*, volume 486 of *Lecture Notes in Computer Science*, pages 213–227. Springer Verlag, 1990.

[63] C. Dwork. On Verification in Secret Sharing. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991,*

Proceedings, volume 576 of *Lecture Notes in Computer Science*, pages 114–128. Springer Verlag, 1991.

[64] P. Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, 27-29 October 1987*, pages 427–437. IEEE Computer Society, 1987.

[65] P. Feldman and S. Micali. Byzantine Agreement in Constant Expected Time (and Trusting No One). In *Proceedings of 26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, 21-23 October 1985*, pages 267–276. IEEE Computer Society, 1985.

[66] P. Feldman and S. Micali. An Optimal Algorithm for Synchronous Byzantine Agreemet. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 639–648. ACM Press, 1988.

[67] P. Feldman and S. Micali. An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. *SIAM Journal of Computing*, 26(4):873–933, 1997.

[68] M. J. Fischer. The Consensus Problem in Unreliable Distributed Systems (A Brief Survey). In M. Karpinski, editor, *Fundamentals of Computation Theory, Proceedings of the 1983 International FCT-Conference, Borgholm, Sweden, August 21-27, 1983*, volume 158 of *Lecture Notes in Computer Science*, pages 127–140. Springer Verlag, 1983.

[69] M. J. Fischer and N. A. Lynch. A Lower Bound on the Time to Assure Interactive Consistency. *Information Processing Letters*, 14(4):183–186, 1982.

[70] M. J. Fischer, N. A. Lynch, and M. Merritt. Easy Impossibility Proofs for Distributed Consensus Problems. In *Fault-Tolerant Distributed Computing*, pages 147–170, 1986.

[71] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM*, 32(2):374–382, 1985.

[72] M. Fitzi. *Generalized Communication and Security Models in Byzantine Agreement*. PhD thesis, ETH Zurich, 2002.

[73] M. Fitzi, J. Garay, S. Gollakota, C. Pandu Rangan, and K. Srinathan. Round-Optimal and Efficient Verifiable Secret Sharing. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 329–342. Springer Verlag, 2006.

[74] M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein, and A. Smith. Detectable Byzantine Agreement Secure against Faulty Majorities. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing, July 21-24, 2002 Monterey, California, USA*, pages 118–126. ACM Press, 2002.

[75] M. Fitzi and M. Hirt. Optimally Efficient Multi-valued Byzantine Agreement. In E. Ruppert and D. Malkhi, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 163–168, 2006.

[76] M. Fitzi, M. Hirt, and U. M. Maurer. Trading Correctness for Privacy in Unconditional Multi-Party Computation (Extended Abstract). In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 121–136. Springer Verlog, 1998.

[77] M. Fitzi, M. Hirt, and U. M. Maurer. General Adversaries in Unconditional Multi-party Computation. In K. Lam, E. Okamoto, and C. Xing, editors, *ASIACRYPT*, volume 1716 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 1999.

[78] M. Fitzi and U. M. Maurer. From Partial Consistency to Global Broadcast. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 494–503. ACM Press, 2000.

[79] M. Fitzi and J. Buus Nielsen. On the Number of Synchronous Rounds Sufficient for Authenticated Byzantine Agreement. In I. Keidar, editor, *Distributed Computing, 23rd International Symposium, DISC 2009, Elche, Spain, September 23-25, 2009. Proceedings*, volume 5805 of *Lecture Notes in Computer Science*, pages 449–463. Springer Verlog, 2009.

[80] M. K. Franklin, M. Gondree, and P. Mohassel. Improved Efficiency for Private Stable Matching. In M. Abe, editor, *Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, February 5-9, 2007, Proceedings*, volume 4377 of *Lecture Notes in Computer Science*, pages 163–177. Springer Verlog, 2006.

[81] M. K. Franklin, M. Gondree, and P. Mohassel. Communication-Efficient Private Protocols for Longest Common Subsequence. In M. Fischlin, editor, *Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings*, volume 5473 of *Lecture Notes in Computer Science*, pages 265–278. Springer Verlog, 2009.

[82] M. K. Franklin and G. Tsudik. Secure Group Barter: Multi-party Fair Exchange with Semi-Trusted Neutral Parties. In R. Hirschfeld, editor, *Financial Cryptography, Second International Conference, FC'98, Anguilla, British West Indies, February 23-25, 1998, Proceedings*, volume 1465 of *Lecture Notes in Computer Science*, pages 90–102. Springer Verlog, 1998.

[83] M. K. Franklin and M. Yung. Communication Complexity of Secure Computation (Extended Abstract). In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing, 4-6 May 1992, Victoria, British Columbia, Canada*, pages 699–710. ACM, 1992.

[84] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. In C. Cachin and J. Camenisch, editors, *Advances*

in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer Verlag, 2004.

[85] Z. Galil, S. Haber, and M. Yung. Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model. In C. Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 135–155. Springer Verlag, 1987.

[86] Z. Galil, A. J. Mayer, and M. Yung. Resolving Message Complexity of Byzantine Agreement and beyond. In *Proceedings of 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 724–733. IEEE Computer Society, 1995.

[87] J. A. Garay and Y. Moses. Fully Polynomial Byzantine Agreement for $n > 3t$ Processors in $t+1$ Rounds. *SIAM Journal of Computing*, 27(1):247–290, 1998.

[88] J. A. Garay and R. Ostrovsky. Almost-Everywhere Secure Computation. In N. P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 307–323. Springer Verlog, 2008.

[89] J. A. Garay and K. J. Perry. A Continuum of Failure Models for Distributed Computing. In A. Segall and S. Zaks, editors, *Distributed Algorithms, 6th International Workshop, WDAG '92, Haifa, Israel, November 2-4, 1992, Proceedings*, volume 647 of *Lecture Notes in Computer Science*, pages 153–165. Springer Verlag, 1992.

[90] R. Genarro. *Theory and Practice of Verifiable Secret Sharing*. PhD thesis, Massachussets Institute of Technilogy, USA, May, 1996.

[91] R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. The Round Complexity of Verifiable Secret Sharing and Secure Multicast. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece. ACM*, pages 580–589. ACM Press, 2001.

[92] R. Gennaro and S. Micali. Verifiable Secret Sharing as Secure Computation. In L. C. Guillou and J. Quisquater, editors, *Advances in Cryptology - EUROCRYPT '95, International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 21-25, 1995, Proceeding*, volume 921 of *Lecture Notes in Computer Science*, pages 168–182. Springer Verlag, 1995.

[93] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, June 28 - July 2, 1998, Puerto Vallarta, Mexico*, pages 101–111. ACM Press, 1998.

[94] O. Goldreich. Secure Multiparty Computation. www.wisdom.weizman.ac.il/~oded/pp.html, 2007.

[95] O. Goldreich, S. Micali, and A. Wigderson. How to Play a Mental Game– A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM Press, 1987.

[96] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing but Their Validity for All Languages in NP have Zero-Knowledge Proof Systems. *Journal of ACM*, 38(3):691–729, 1991.

[97] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Secret Sharing Or: How to Cope with Perpetual Leakage. In D. Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352. Springer Verlag, 1995.

[98] M. Hirt, U. Maurer, and B. Przydatek. Efficient Secure Multiparty Computation. In T. Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 143–161. Springer Verlag, 2000.

[99] M. Hirt and U. M. Maurer. Complete Characterization of Adversaries Tolerable in Secure Multi-Party Computation. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, California, USA, August 21-24, 1997*, pages 25–34. ACM Press, 1997.

[100] M. Hirt and U. M. Maurer. Player Simulation and General Adversary Structures in Perfect Multiparty Computation. *Journal of Cryptology*, 13(1):31–60, 2000.

[101] M. Hirt and U. M. Maurer. Robustness for Free in Unconditional Multiparty Computation. In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 101–118. Springer Verlag, 2001.

[102] M. Hirt, U. M. Maurer, and V. Zikas. MPC vs. SFE : Unconditional and Computational Security. In J. Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, volume 5350 of *Lecture Notes in Computer Science*, pages 1–18. Springer Verlag, 2008.

[103] M. Hirt and J. B. Nielsen. Upper Bounds on the Communication Complexity of Optimally Resilient Cryptographic Multiparty Computation. In B. K. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and*

Information Security, Chennai, India, December 4-8, 2005, Proceedings, volume 3788 of *Lecture Notes in Computer Science*, pages 79–99. Springer Verlag, 2005.

[104] M. Hirt and J. B. Nielsen. Robust Multiparty Computation with Linear Communication Complexity. In C. Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 463–482. Springer Verlag, 2006.

[105] M. Hirt, J. B Nielsen, and B. Przydatek. Cryptographic Asynchronous Multi-party Computation with Optimal Resilience (Extended Abstract). In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 322–340. Springer Verlag, 2005.

[106] M. Hirt, J. B Nielsen, and B. Przydatek. Asynchronous Multi-Party Computation with Quadratic Communication. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 473–485. Springer Verlag, 2008.

[107] Z. Huang, W. Qiu, Q. Li, and K. Chen. Efficient Secure Multiparty Computation Protocol in Asynchronous Network. In J. H. Park, H. Chen, M. Atiquzzaman, C. Lee, T. Kim, and S. Yeo, editors, *Proceedings of Advances in Information Security and Assurance, Third International Conference and Workshops, ISA 2009, Seoul, Korea*, volume 5576 of *Lecture Notes in Computer Science*, pages 152–158. Springer Verlag, June 2009.

[108] Y. Ishai and E. Kushilevitz. Randomizing Polynomials: A New Representation with Applications to Round-Efficient Secure Computation. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California*, pages 294–304. IEEE Computer Society, 2000.

[109] J. Katz, C. Koo, and R. Kumaresan. Improving the Round Complexity of VSS in Point-to-Point Networks. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 499–510. Springer Verlag, 2008.

[110] J. Katz and C. Y. Koo. On Expected Constant-Round Protocols for Byzantine Agreement. In C. Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara,*

California, USA, August 20-24, 2006, Proceedings, Lecture Notes in Computer Science, pages 445–462. Springer Verlag, 2006.

[111] J. Katz and C. Y. Koo. Round-Efficient Secure Computation in Point-to-Point Networks. In M. Naor, editor, *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*, pages 311–328. Springer Verlag, 2007.

[112] J. Katz, R. Kumaresan, A. Choudhary, S. Narayanan, A. Patra, A. Raghunathan, and C. Pandu Rangan. The Round Complexity of Verifiable Secret Sharing: The Statistical Case. Manuscript, 2010.

[113] L. Kissner and D. X. Song. Privacy-Preserving Set Operations. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257. Springer Verlag, 2005.

[114] L. Lamport. The Weak Byzantine Generals Problem. *Journal of ACM*, 30(3):668–676, 1983.

[115] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[116] R. Li and C. Wu. An Unconditionally Secure Protocol for Multi-Party Set Intersection. In J. Katz and M. Yung, editors, *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*, volume 4521 of *Lecture Notes in Computer Science*, pages 222–236. Springer Verlag, 2007.

[117] Y. Lindell, A. Lysyanskaya, and T. Rabin. On the Composition of Authenticated Byzantine Agreement. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montral, Qubec, Canada*, pages 514–523. ACM Press, 2002.

[118] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[119] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. North-Holland Publishing Company, 1978.

[120] U. M. Maurer. Secure Multi-party Computation made Simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.

[121] R. J. McEliece and D. V. Sarwate. On Sharing Secrets and Reed-Solomon Codes. *Communications of the ACM*, 24(9):583–584, 1981.

[122] Y. Moses and O. Waarts. Coordinated Traversal: $(t+1)$-Round Byzantine Agreement in Polynomial Time. *Journal of Algorithms*, 17(1):110–156, 1994.

[123] W. Ogata and K. Kurosawa. Optimum Secret Sharing Scheme Secure against Cheating. In U. M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 200–211. Springer Verlag, 1996.

[124] R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks. In *Proceedings of the Tenth Annual ACM Symposium on Princiles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, 1991*, pages 51–61. ACM Press, 1991.

[125] A. Patra, A. Choudhary, T. Rabin, and C. Pandu Rangan. The Round Complexity of Verifiable Secret Sharing Revisited. In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 487–504. Springer Verlag, 2009.

[126] A. Patra, A. Choudhary, and C. Pandu Rangan. Round Efficient Unconditionally Secure Multiparty Computation Protocol. In D. R. Chowdhury, V. Rijmen, and A. Das, editors, *Progress in Cryptology - INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings*, volume 5365 of *Lecture Notes in Computer Science*, pages 185–199. Springer Verlag, 2008.

[127] A. Patra, A. Choudhary, and C. Pandu Rangan. Efficient Asynchronous Byzantine Agreement with Optimal Resilience. Submitted to Distributed Computing Journal. A preliminary version of this article appeared in Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing, PODC 2009, Calgary, Alberta, Canada, August 10-12, pages 92-101, 2009.

[128] A. Patra, A. Choudhary, and C. Pandu Rangan. Efficient Statistical Asynchronous Verifiable Secret Sharing with Optimal Resilience. In K. Kurosawa, editor, *Information Theoretic Security, Fourth International Conference, ICITS 2009, Shizuoka, Japan, December 3-6, 2009, Proceedings*, volume 5973 of *Lecture Notes in Computer Science*. Springer Verlag, 2009.

[129] A. Patra, A. Choudhary, and C. Pandu Rangan. Information Theoretically Secure Multi Party Set Intersection Re-visited. In M. J. Jacobson Jr., V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*. Springer Verlag, 2009.

[130] A. Patra, A. Choudhary, and C. Pandu Rangan. Round Efficient Unconditionally Secure MPC and Multiparty Set Intersection with Optimal Resilience. In *Progress in Cryptology - INDOCRYPT 2009, 10th International Conference on Cryptology in India, New Delhi, India, December 13-16, 2009. Proceedings*, volume 5922 of *Lecture Notes in Computer Science*, pages 398–417. Springer Verlog, 2009.

[131] A. Patra, A. Choudhary, and C. Pandu Rangan. Communication Efficient Perfectly Secure VSS and MPC in Asynchronous Networks with Optimal Resilience. In D.J. Bernstein and T. Lange, editors, *Advances in Cryptology - AFRICACRYPT'10, Third International Conference in Cryptology in Africa, Stellenbosch, South Africa, May 3-6, 2009, Proceedings*, volume 6055 of *Lecture Notes in Computer Science*, pages 184–202. Springer Verlag, 2010.

[132] M. Pease, R. E. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of ACM*, 27(2):228–234, 1980.

[133] T. Pedersen. Non-interactive and Information-theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology - CRYPTO '91, Santa Barbara, California, USA, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 1991.

[134] B. Pfitzmann and M. Waidner. Unconditional Byzantine Agreement for any number of faulty processors. In A. Finkel and M. Jantzen, editors, *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, volume 577 of *Lecture Notes in Computer Science*, pages 339–350. Springer Verlag, 1992.

[135] B. Prabhu, K. Srinathan, and C. Pandu Rangan. Trading Players for Efficiency in Unconditional Multiparty Computation. In S. Cimato, C. Galdi, and G. Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*, volume 2576 of *Lecture Notes in Computer Science*, pages 342–353. Springer Verlag, 2002.

[136] M. O. Rabin. Randomized Byzantine Generals. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto California, 3-5 November 1993*, pages 403–409. IEEE Computer Society, 1983.

[137] T. Rabin. Robust Sharing of Secrets when the Dealer is Honest or Cheating. *Journal of ACM*, 41(6):1089–1109, 1994.

[138] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 73–85. ACM Press, 1989.

[139] D. V. S. Ravikant, M. Venkitasubramaniam, V. Srikanth, K. Srinathan, and C. Pandu Rangan. On Byzantine Agreement over (2, 3)-Uniform Hypergraphs. In R. Guerraoui, editor, *Distributed Computing, 18th International Conference, DISC 2004, Amsterdam, The Netherlands, October 4-7, 2004, Proceedings*, volume 3274 of *Lecture Notes in Computer Science*, pages 450–464. Springer Verlog, 2004.

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

[141] K. Srinathan, A. Narayanan, and C. Pandu Rangan. Optimal Perfectly Secure Message Transmission. In M. K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference,*

Santa Barbara, California, USA, August 15-19, 2004, Proceedings, volume 3152 of *Lecture Notes in Computer Science*, pages 545–561. Springer Verlag, 2004.

[142] K. Srinathan, A. Patra, A. Choudhary, and C. Pandu Rangan. Unconditionally Secure Message Transmission in Arbitrary Directed Synchronous Networks Tolerating Generalized Mixed Adversary. In W. Li, W. Susilo, U. K. Tupakula, R. Safavi-Naini, and V. Varadharajan, editors, *Proceedings of the 2009 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009, Sydney, Australia, March 10-12, 2009*, pages 171–182. ACM, 2009.

[143] K. Srinathan and C. Pandu Rangan. Efficient Asynchronous Secure Multiparty Distributed Computation. In B. K. Roy and E. Okamoto, editors, *Progress in Cryptology - INDOCRYPT 2000, First International Conference in Cryptology in India, Calcutta, India, December 10-13, 2000, Proceedings*, volume 1977 of *Lecture Notes in Computer Science*, pages 117–129. Springer Verlag, 2000.

[144] K. Srinathan and C. Pandu Rangan. Possibility and Complexity of Probabilistic Reliable Communication in Directed Networks. In E. Ruppert and D. Malkhi, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 265–274. ACM, 2006.

[145] M. Stadler. Publicly Verifiable Secret Sharing. In U. M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 190–199. Springer Verlag, 1996.

[146] M. Tompa and H. Woll. How to Share a Secret with Cheaters. In A. M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 261–265. Springer Verlag, 1986.

[147] M. Tompa and H. Woll. How to Share a Secret with Cheaters. *Journal of Cryptology*, 1(2):133–138, 1988.

[148] S. Toueg. Randomized Byzantine Agreements. In *Proceedings of the Third Annual ACM Symposium on Princiles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 163–178. ACM Press, 1984.

[149] S. Toueg, K. J. Perry, and T. K. Srikanth. Fast Distributed Agreement. *SIAM Journal of Computing*, 16(3):445–457, 1987.

[150] R. Turpin and B. A. Coan. Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement. *Information Processing Letters*, 18(2):73–76, 1984.

[151] A. C. Yao. Protocols for Secure Computations. In *Proceedings of 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.

[152] H. Zheng, G. Zheng, and L. Qiang. Batch Secret Sharing for Secure Multiparty Computation in Asynchronous Network. *Journal of Shanghai Jiaotong Univ. (Sci.)*, 14(1):112–116, 2009.

[153] V. Zikas, S. Hauser, and U. M. Maurer. Realistic Failures in Secure Multiparty Computation. In O. Reingold, editor, *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, volume 5444 of *Lecture Notes in Computer Science*, pages 274–293. Springer Verlag, 2009.