On the Round Complexity Landscape of Secure Multi-party Computation

A THESIS

SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

IN THE

Faculty of Engineering

BY

Divya Ravi

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

June, 2020

Declaration of Originality

I, **Divya Ravi**, with SR No. **04-04-00-10-12-16-1-13995** hereby declare that the material presented in the thesis titled

On the Round Complexity Landscape of Secure Multi-party Computation

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2016-2020**.
With my signature, I certify that:

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

Date: June 6, 2020 Student Signature

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

Advisor Name: Arpita Patra Advisor Signature

DEDICATED TO

My wonderful family -

Amma, Appa and my sister Tara

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Dr. Arpita Patra for introducing me to research in Computer Science and this amazing field of Secure Multi-party Computation. My research journey started with my Master's project under her supervision and her continuous guidance and motivation was a primary factor in driving me to pursue a Ph.D. Her passion for research is truly contagious and commendable. I consider myself extremely fortunate to have got this opportunity to work with her closely which enabled me to discover and enjoy the process of research. Her strive for excellence has always been inspiring and I am grateful to her for her immense support and faith. I will always cherish being one of the first students she has supervised and thoroughly enjoyed not only the technical but all the random conversations and discussions we have had over the past few years.

Besides my advisor, I would like to express my sincere thanks to Dr. Ashish Choudhury, Professor at IIIT Bangalore for his generous support and warm encouragement throughout. Ever since he mentored and guided me through my first couple of projects, he has been a constant source of inspiration to me for his genuine kindness, warmth and strong work ethic. I am deeply grateful to him for his invaluable suggestions and constructive comments. During my studies, I got great opportunities to visit some of the best researchers in the cryptography community at Aarhus university, Cornell-Tech and Technion-Israel Institute of Technology. I owe my deepest gratitude to Prof. Ivan Damgard, the Aarhus Crypto Group, Prof. Yuval Ishai and Prof. Muthuramakrishnan Venkitasubramaniam for hosting me and giving me the chance to interact with their group. These internships have played a key role in boosting my confidence and my interactions with them have helped me develop new insights and expand my research horizon. These opportunities enabled me to engage in collaborative projects with Prof. Yuval Ishai, Dr. Sikhar Patranabis, Dr. Abhishek Jain, Aarushi Goel, Prof. Ivan Damgard and Daniel Escudero. I have learnt a lot from each of them and would like to express my heartfelt gratitude for the fruitful discussions. I am also sincerely grateful to Dr. Pratyay Mukherjee, Dr. Peihan Miao and Dr. Saikrishna Badrinarayan for giving me the opportunity to work with them at Visa Research as a spring intern and for the interesting ongoing research discussions.

Acknowledgements

I would like to offer my special heartfelt thanks to my labmates at "Cryptography and Information Security (CrIS)" lab at IISc. The positive and fun working environment made my research experience truly joyful. I will genuinely miss the endless discussions and numerous outings with my labmates Ajith, Nishat, Protik, Mahak, Harsh and Shravani. I am particularly grateful to Ajith whose support and honest feedback to all the members keeps the lab glued together. This list is incomplete without mentioning my former colleagues Swati, Megha and Pratik with whom I have had the most memorable and enjoyable collaborations. I extend my deep gratitude to the student co-authors of some of the works constituting this thesis - Swati, Megha and Arun. I would also like to thank some of the visitors of our lab - Prof. Chaya Ganesh, Prof. Carmit Hazay, Yashvanth Kondi, Gayathri Garimella, Laasya Bangalore, Rahul Raichuri and Rishabh Bhadaurai for sharing their knowledge and the fun discussions.

My sincere acknowledgments to IACR/RWC, Microsoft Research India, IARCS, Tata Trusts, CSA departmental fund for women PhD students (sponsored by Prof. Priti Shankar's family) and the institute GARP sponsporship for supporting my travel to international conferences. I have received generous support and help from faculty and office staff of Department of Computer Science and Automation (CSA), IISc throughout my M.E and Ph.D course of study, for which I am immensely grateful. I would like to offer special thanks to Prof. Narsimhan Murthy, Prof. Narahari, Prof. Shalabh Bhatnagar, Prof. Bhavana Kanukurthi and Prof. Vinod Ganapathy for their warm encouragement and constructive feedback. The opportunities and the encouraging environment of the institute has truly made my journey at IISc, an invaluable learning experience.

My experience at IISc has been beautiful due to some amazing people I met here during the course of Masters and Ph.D. Starting with my biggest cheerleader Antareep who has been incredibly supportive and my rock; I would like to express my heartfelt thanks to Anupam, Anwesha, Sruthi, Vishal, Jay, Ramanpreet, Anjali, Hari Om and Manohar for making my stay at IISc memorable. Lastly but most importantly, I would like to extend immense love to my wonderful family - Amma, Appa and my sister Tara. Their endearing enthusiasm about every small and big milestone in this journey, be it any presentation or travel kept me going. Their unconditional moral support and affection means the world to me and I dedicate this thesis to them.

Abstract

Secure multi-party computation (MPC) allows a group of n mutually distrustful parties to jointly perform a computation on their private inputs in a secure way, so that no adversary $\mathcal{A}$ actively corrupting a subset of the parties can learn more information than their outputs (*privacy*), nor can they affect the outputs of the computation other than by choosing their own inputs (*correctness*). The round complexity of MPC protocols is a fundamental question in the area of secure computation and its study constitutes a phenomenal body of work in the MPC literature. The research goal of this thesis is to advance the state of the art by expanding this study of round complexity to various realistic adversarial settings and network models. The questions addressed in the thesis are of both theoretical and practical importance.

The first part of the thesis studies round-optimal (more generally, round-efficient) MPC protocols for small population, namely involving 3 (3PC) and 4 (4PC) parties tolerating single active corruption (honest majority). We address two broad categories of questions -

- We settle the exact round complexity of 3PC in honest-majority setting, for a range of security notions such as selective abort (sa), unanimous abort (ua), fairness (fn) and guaranteed output delivery (god). sa, the weakest in the lot, allows the corrupt parties to selectively deprive some of the honest parties of the output. In the mildly stronger version of ua, either all or none of the honest parties receive the output. fn implies that the corrupted parties receive their output only if all honest parties receive output and lastly, the strongest notion of god implies that the corrupted parties cannot prevent honest parties from receiving their output. We focus on two network settings– pairwise-private channels without and with a broadcast channel.

- On the more practical side, we present efficient, constant-round 3PC and 4PC protocols in the honest-majority setting that achieve strong security notions of fn and god. Being constant-round and striking a good balance between the complexity measures of communication, computation and round complexity, our constructions are suitable for high-latency networks such as the Internet.

Abstract

The second part of the thesis extends the study of round complexity beyond the traditional settings and towards more realistic adversarial settings. Our contributions are:

- The two traditional streams of MPC protocols consist of– (a) protocols achieving god or fn in the honest-majority setting and (b) protocols achieving (ua, sa) in the dishonest-majority setting. The favorable presence of honest majority amongst the participants is necessary to achieve the stronger notions of god or fn[65]. Unfortunately, a protocol in *one* setting completely breaks down in the *other* setting. We overcome this demarcation of study of round complexity of MPC based on resilience (i.e honest majority or dishonest majority) and explore round complexity for an interesting class of protocols called the Best-of-both-Worlds (BoBW) MPC which simultaneously achieve fn / god in honest majority and ua in dishonest majority. We nearly settle the question of exact round complexity of BoBW protocols under the assumption of no setup (plain model), public setup (common random / reference string a.k.a CRS) and private setup (public-key infrastructure a.k.a PKI).

- In a generalised adversarial setting where the adversary is allowed to corrupt both passively (corrupt parties follow the protocol specifications but the adversary learns the internal state) and actively (corrupt parties deviate arbitrarily from the protocol), the necessary bound for a n-party fair or robust (achieving god) protocol turns out to be $t_a + t_p < n$, where t_a, t_p denote the threshold for active and passive corruption with the latter subsuming the former. Subsuming the traditional settings as boundary special cases, we study the dynamic corruption setting which opens up a range of possible corruption scenarios for the adversary. While dynamic corruption includes the entire range of thresholds for (t_a, t_p) starting from $(\lceil \frac{n}{2} \rceil - 1, \lfloor n/2 \rfloor)$ to $(0, n-1)$, the boundary corruption restricts the adversary only to the boundary cases of $(\lceil \frac{n}{2} \rceil - 1, \lfloor n/2 \rfloor)$ and $(0, n-1)$. We overcome the demarcation of study of round complexity of MPC based on single type of corruption (i.e passive or active) and settle the exact round complexity of fair and robust MPC against dynamic and boundary adversaries under the assumption of public setup (CRS).

While the above two parts include results in the computational (assumes polynomially-bounded adversaries) and fully synchronous setting (assumes network channels with bounded-delay), the final part of the thesis involves information-theoretic setting (tolerates computationally unbounded adversaries) and introduces asynchrony in the network as well. We address the following fundamental questions wrt MPC and VSS (Verifiable Secret Sharing, which is

a fundamental building block for many distributed cryptographic tasks including MPC). Our main contribution is as follows:

- Perfectly-secure (information-theoretic with no error) VSS and MPC protocols in asynchronous network (allows arbitrary network delays) tolerate only at most one-fourth of corruption, while their counterparts in synchronous network sustain against at most one-third corruption. Moreover property-wise, synchronous protocols provide much stronger guarantees than the asynchronous counterparts. Taking note of the fact that asynchronous network is more realistic on one hand and on the other, synchrony of a network has positive impact on several aspects of distributed protocols including properties and fault-tolerance, we explore the power of *hybrid* networks that combines best of both the worlds by supporting a few synchronous rounds at the onset of a protocol execution, before turning to asynchronous mode. In hybrid networks, we investigate various feasibility questions pertaining to protocols giving guarantees attainable in synchronous as well as asynchronous networks. Specifically, we wish to add and find the minimum synchrony assumption needed. For asynchronous protocols, we wish to bridge the fault-tolerance gap between synchronous and asynchronous protocols with minimum synchrony assumption needed, leveraging the initial synchronous rounds. For synchronous protocols, we explore if the known lower bounds on round complexity can be circumvented, leveraging the asynchronous phase available in the hybrid network.

Publications based on this Thesis

Accepted Papers

[**C1**] Patra, A., **Ravi, D.**: *On the exact round complexity of secure three-party computation.* In: Shacham, H., Boldyreva, A. (eds.) *CRYPTO 2018*, Part II. LNCS, vol. 10992, pp. 425–458. Springer, Cham (2018).

[**C2**] Byali, M., Joseph, A., Patra, A., **Ravi, D.**.: *Fast secure computation for small population over the internet.* In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, *CCS 2018*, Toronto, ON, Canada, October 15-19, 2018. pp. 677–694 (2018).

[**C3**] A. Patra, **Ravi, D.**. *Beyond honest majority: The round complexity of fair and robust multi-party computation.* In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019*, Part I, volume 11921 of LNCS, pages 456–487. Springer, Heidelberg, 2019.

[**J1**] A. Patra, **Ravi, D.**, *On the power of hybrid networks in multi-party computation*, *IEEE Trans. Inf. Theory*, vol. 64, no. 6, pp. 4207-4227, Jun. 2018.

Papers in Submission

[**C4**] Patra, A., **Ravi, D.**, Singla, S. *On the Exact Round Complexity of Best-of-Both-Worlds Multi-party Computation*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Secure Multi-party Computation (MPC)[182, 107, 55, 30, 56], arguably regarded as the "holy grail" of cryptography, allows a group of n mutually distrustful parties to jointly perform a computation on their private inputs in a secure way, so that no adversary $\mathcal{A}$ actively corrupting a subset of the parties can learn more information than their outputs (*privacy*), nor can they affect the outputs of the computation other than by choosing their own inputs (*correctness*). In the presence of a trusted third party (TTP), the above problem is trivial since all parties could privately send their inputs to the TTP, which would compute the desired function and subsequently return the output to the parties. Intuitively, an MPC protocol replaces the TTP and guarantees the same level of security that the TTP provides.

The field of MPC originated with the seminal work of [182] which introduced the classical "Yao's millionaire problem" (the question of how two millionaires can determine who is richer while keeping their actual wealth private). Since then, MPC has evolved into an active area of research with a rich body of work comprising of both theoretical and practical achievements. Its study is motivated not only by the fact that it gives us a general framework to study cryptography (as most cryptographic tasks can be casted as secure computation problems) but also since it enables various privacy-preserving applications such as secure auctions [41], secure machine learning [158, 155, 157, 53], secure benchmarking [77], statistical data analysis [40], email-filtering [144], financial data analysis [40] and privacy-preserving data mining [149]. In more detail, a motivating example of a real-life scenario that demands privacy-preserving computation is the following - Consider an airline company that has a private database of its list of passengers and the government which owns a database with sensitive information of blacklisted passengers. It is of mutual interest to both these entities to find the intersection of their individual databases without compromising on privacy, which is enabled by MPC. MPC has been studied extensively in various settings that explore different kinds of adversaries, setup

and network models.

1.1 Dimensions of MPC

The various facets of real-life computing environments are captured by the fundamental dimensions of MPC which includes the underlying communication network, type of adversary and the setup to name a few. We outline the most prominent dimensions below. Looking ahead, throughout this thesis, we use various combinations of the below mentioned models of network, adversary and setup.

1.1.1 Network Model

The standard network model in MPC is the *complete* network which assumes that every pair of distinct parties is directly connected by point-to-point secure and authentic channels. Sometimes, the presence of a broadcast channel is additionally assumed; where a broadcast channel allows any party to send a message identically to all other parties in the network. An important network dimension is *synchrony* based on which there are three categories:

- **Synchronous:** In the synchronous setting, it is assumed that all parties have access to a common global clock and the delay of messages in the channels of the network is bounded by a known constant. This allows protocols to proceed in rounds, with the strong delivery guarantee that every message sent in any given round is delivered to all the recipients in the same round.

- **Asynchronous:** In contrast to the above, in the asynchronous setting, there is no global clock. It is assumed that the channels in the network may have arbitrary delays and may deliver messages in any arbitrary order, with the only restriction that every sent message must eventually be delivered. In order to model the worst case, the adversary is allowed to control the scheduling of messages in the network.

- **Hybrid:** A network that is asynchronous in nature and yet supports a few synchronous rounds at the onset of a protocol execution is denoted as hybrid network [23, 167]. While we consider this notion, an alternative notion of hybrid networks appears in [75] where a synchronization point is considered (the network is asynchronous before and after the point).

1.1.2 Adversarial Model

Throughout this thesis, we consider a *static* and *threshold* adversary; where the former assumes that the adversary decides on the set of parties it would corrupt before the protocol begins

(as opposed to the adaptive model where the adversary is allowed to choose which parties to corrupt during the protocol execution) and the latter assumes that the number of corrupt parties is bounded by a threshold t (as opposed to a non-threshold where an adversary structure comprising a set of subset of the parties is defined, among which the adversary can corrupt one of the subsets). Following is the discussion on the most relevant dimensions related to the adversarial model.

1.1.2.1 Computational Power

Based on the computational power of the adversary, the two main categories are the **information-theoretic** and **cryptographic** settings. While the former assumes that the adversary may have unbounded computing power, the latter assumes a PPT adversary i.e bounded by probabilistic polynomial time. The information-theoretic setting further has two categories of protocols - **(a)** *perfectly-secure* (tolerate no error) and **(b)** *statistically-secure* (tolerate negligible error probability, where a negligible function is considered to be one that grows slower than any inverse polynomial; details in Chapter 2).

1.1.2.2 Type of Misbehaviour / Corruption

Based on the allowed misbehaviour, an adversary can be categorized into the following types:

- **Passive / Semi-honest**: The corrupt parties in this adversarial model are honest-but-curious i.e they follow the protocol specifications but the adversary learns the internal state of the corrupt parties which it may use to learn more information (i.e more than what is allowed as per the security guarantees of the protocol).
- **Active / Malicious**: In this setting, the adversary exercises total control over the corrupt parties who may deviate from the protocol steps in any arbitrary manner.
- **Mixed**: This is a generalized adversarial model where the adversary may simultaneously perform both types of corruption i.e he may corrupt a subset of parties actively, and additionally corrupt few others passively.

1.1.3 Setup

Based on the type of setup assumed, there are three well-studied models in the MPC literature - **(a) plain model** which does not assume any kind of trusted setup, **(b) public setup**, also referred to as the *CRS model* which assumes that a trusted common random / reference string (CRS) is available to the parties and **(c) private setup**, also referred to as the *PKI model* which assumes that parties have access to a public-key infrastructure (sometimes in addition to access to a CRS).

1.2 Attributes of MPC

While the setting of an MPC protocol is defined by the above discussed dimensions of adversarial, network and the setup models; the analysis and comparison of protocols in the same setting is done by means of the following attributes:

- **Resilience:** This is a measure of the number of corrupt parties that can be tolerated i.e the defined threshold t. Among the protocols in the same setting and providing the same security guarantees, the one having higher resilience is considered preferable. The most common categorization based on resilience is the *honest majority* ($t < n/2$) and the *dishonest majority* ($t < n$) settings.

- **Quality / Degree of Robustness:** Based on the degree of robustness, the categorization is as follows (starting from the strongest i.e the most desirable to the weakest i.e the least preferred)

  - *Guaranteed output delivery* (god): The adversary cannot prevent honest parties from receiving their output. This is the most desirable security notion. We refer to such protocols as being *robust*.
  - *Fairness* (fn): The adversary obtains the output if and only if the honest parties do. In other words, either all or none of the parties receive output.
  - *Unanimous Abort* (ua): Either all or none of the honest parties obtain the output. Here, the protocols may be unfair i.e the adversary may get the output while the honest parties don't but there is an agreement amongst the honest parties with respect to the output.
  - *Selective Abort* (sa): This is the weakest security notion where the adversary may selectively deprive a subset of the honest parties of the output.

  Other notions include *identifiable abort* (idua) and *identifiable fairness* (idfair) where protocols achieving ua and fn respectively satisfy the following useful identifiability property: if the parties do not receive the function output, then every party learns the identity of atleast one corrupted party.

- **Complexity Measures:** The complexity of an MPC protocol is measured in terms of the following fundamental parameters:

- *Round Complexity*: For MPC protocols in the synchronous network, the round complexity is a measure of the number of rounds i.e the number of sequential interactions in the protocol execution.
- *Communication Complexity*: This is measured as the total number of bits communicated by the honest parties in the protocol.
- *Computation Complexity*: This captures the computational resources utilized by the parties during the protocol executions. More concretely, this can be measured in terms of the number and type of mathematical operations, running time etc.

The lesser the number of rounds / bits / computation involved in a protocol, the more round / communication / computation - efficient it is considered as being. Looking ahead, the focus of this thesis is the round complexity of MPC in various settings.

Before moving on to the contributions of the thesis related to round complexity of MPC under various settings, we outline the relevant literature below.

1.3 Related Work on Round Complexity of MPC

The phenomenal body of work done on round complexity catering to various adversarial settings and network models emphasises its theoretical importance and practical relevance. For instance, the exact round complexity of MPC independently in honest and dishonest majority has been examined and the recent literature is awash with a bunch of upper bounds that eluded for quite a long time [93, 35, 113, 15]. We review the round complexity of the honest-majority and dishonest-majority MPC in the *computational* setting below as it is most relevant to the contributions in this thesis. To begin with, 2 rounds are known to be necessary to realize any MPC protocol, regardless of the setting, no matter whether a setup is assumed or not as long as the setup (when assumed) is independent of the inputs of the involved parties. This is because in a 1-round protocol, a corrupt party could repeatedly evaluate the "residual function" with the inputs of the honest parties fixed on many different inputs of its own (referred as "residual function" attack) [112].

Dishonest Majority. When no setup is assumed (plain model), 5 rounds are known to be necessary in non-simultaneous message model for actively-secure 2PC [136]. This bound can be improved to 4 even for the general case of dishonest majority in simultaneous message model [95]. Tight upper bounds appear in [44, 3, 64, 113, 15, 60], with the latter three presenting constructions under polynomial-time assumptions. In the presence of a public setup (Common Reference String a.k.a. CRS setting), the lower bound comes down to 2 rounds [112]. A series of

work present matching upper bounds under various assumptions [94, 160, 92], culminating with the works of [93, 35] that attain the goal under the minimal assumption of 2-round oblivious transfer (OT).

Honest Majority. In the honest majority setting which is shown to be necessary [65] and sufficient [30, 56, 66] for the feasibility of protocols with fn (fairness) and god (guaranteed output delivery), the study on round complexity has seen the following interesting results. In the plain model, 3 rounds are shown to be necessary for fn (hence for god) protocols, in the presence of pairwise-private and broadcast channels for $t \geq 2$ active corruptions [102] and for any t as long as $n/3 \leq t < n/2$ [166]. Circumventing this 3-round lower bound for fn, [129, 126] show 2-round protocols for $n \geq 4$ against a single active corruption achieving god even without a broadcast channel. The matching upper bounds in the plain model appear in [4] for the general case under public-key assumption, [16] based on threshold multi-key FHE (fully-homomorphic encryption) and in [166] for the special case of 3PC under the minimal assumption of (injective) one-way functions (OWF). In the CRS model, 3 rounds remains to be the lower bound for fn in a setting where broadcast is the only medium of communication (broadcast-only setting) [108] and additionally with point-to-point channels [166, 102, 168]. Given PKI, the bound can be improved to 2 [108].

1.4 The Contribution of this Thesis

The research goal of this thesis is to advance the state of the art by expanding the scope of the existing study of round complexity (outlined above in Section 1.3) to various realistic adversarial settings and network models. The questions addressed in the thesis are of both theoretical and practical importance. We establish new lower bounds on round complexity of MPC in different settings and present matching upper bound constructions. Most of our upper bound constructions are based on garbled circuits (referred to as GC, elaborated in Chapter 2), which is a celebrated and standard technique to construct MPC protocols and constitutes the basis of numerous constructions in the MPC literature. Following are our contributions:

1.4.1 MPC for small population

We study round-optimal (more generally, round-efficient) MPC protocols for small population, namely involving 3 (3PC) and 4 (4PC) parties tolerating single active corruption (honest majority). MPC with small number of parties maintaining an honest majority make a fascinating area of research due to myriad reasons as highlighted below. First, they present useful use-cases in practice, as it seems that the most likely scenarios for secure MPC in practice would involve a small number of parties. In fact, the first large scale implementation of secure MPC, namely the

Danish sugar beet auction [41] was designed for the three-party setting. Several other applications solved via 3PC / 4PC include statistical data analysis [40], email-filtering [144], financial data analysis [40], distributed credential encryption service [159] and secure machine learning [158, 155, 157, 53, 171, 47]. The practical efficiency of these protocols has thus got considerable emphasis and some of them have evolved to technologies [96, 39, 143, 144, 57, 91, 7]. Second, in practical deployments of secure computation between multiple servers that may involve long-term sensitive information, three or more servers are preferred as opposed to two. This enables recovery from faults in case one of the servers malfunctions. Third and importantly, practical applications usually demand strong security goals such as fn and god which are feasible *only* in honest majority setting [65].

Driven by the above motivation, we address two broad categories of questions - **(1)** First, we settle the exact round complexity of 3PC in honest-majority setting, for a range of security notions such as selective abort (sa), unanimous abort (ua), fairness (fn) and guaranteed output delivery (god). We focus on two network settings– pairwise-private channels without and with a broadcast channel. **(2)** On the more practical side, we present efficient, constant-round 3PC and 4PC protocols in the honest-majority setting that achieve strong security notions of fn and god. Being constant-round and striking a good balance between the complexity measures of communication, computation and round complexity, our constructions are suitable for high-latency networks such as the Internet. We elaborate on the results below.

1.4.1.1 On the Exact Round Complexity of 3PC

We set our focus on the exact round complexity of 3PC protocols with one active corruption in the *plain model* achieving a range of security notions, namely sa, ua, fn and god in a setting with pair-wise private channels and without or with a broadcast channel. In the minimal setting of pair-wise private channels, it is known that 3PC with sa is feasible in just two rounds [129], while god is infeasible to achieve irrespective of the number of rounds [67]. No bound on round complexity is known for ua or fn. In the setting with a broadcast channel, the result of [159] implies 3-round 3PC with ua. Neither the round optimality of the [159] construction, nor any bound on round complexity is known for protocols with fn and god.

We settle all the above questions via two lower bound results and three upper bounds. Both our lower-bounds extend for general n and t with strict honest majority i.e. $n/3 \leq t < n/2$ and hold even in the CRS model [168]. They imply tightness of several known constructions of [129] and complement the lower bound of [102] which holds for only $t > 1$. Our upper bounds are from injective (one-to-one) one-way functions (referred to as OWF, one-way function is a function that is easy to compute on every input, but hard to invert given the image of a random

input; elaborated in Chapter 2). The fundamental concept of garbled circuits (GC) contributes as their key basis, following several prior works in this domain [58, 129, 159]. The techniques in our upper bounds do not seem to extend for $t > 1$, leaving open designing round-optimal protocols for the general case with various security notions (with minimal assumptions).

Without Broadcast Channel. We show that three rounds are necessary to achieve 3PC with ua and fn, in the absence of a broadcast channel. The sufficiency is proved via a 3-round fair protocol (which also achieves ua security). Our lower bound result immediately implies tightness of the 3PC protocol of [129] achieving sa in two rounds, in terms of security achieved. This completely settles the questions on exact round complexity of 3PC in the minimal setting of pair-wise private channels.

With Broadcast Channel. With access to a broadcast channel, we show that it takes just two rounds to get 3PC with ua, implying non-optimality of the 3-round construction of [159]. On the other hand, we show that three rounds are necessary to construct a 3PC protocol with fn and god. The sufficiency for fn already follows from our 3-round fair protocol without broadcast. The sufficiency for god is shown via yet another construction in the presence of broadcast. The lower bound result restricted for $t = 1$ complements the lower bound of [102] making three rounds necessary for MPC with fn in the honest majority setting for all the values of t. The lower bound further implies that for two-round fair (or robust i.e achieving god) protocols with one corruption, the number of parties needs to be at least four, making the 4PC protocol of [129] an optimal one. Notably, our result does not contradict with the two-round protocol of [108] that assumes PKI (where the infrastructure contains the public keys of a 'special' FHE), CRS and also broadcast channel.

The above results on exact round complexity of 3PC appeared in [166] (for full version, refer [168]). The table below captures the complete picture of the round complexity of 3PC. Notably, broadcast facility only impacts the round complexity of ua and god, leaving the round complexity of sa and fn unperturbed.

Security	Without Broadcast	References Necessity/Sufficiency	With Broadcast	References Necessity/Sufficiency
Selective Abort (sa)	2	[112] / [129]	2	[112] / [129]
Unanimous Abort (ua)	3	**Our Work [166]** / **Our Work [166]**	2	[112] / **Our Work [166]**
Fairness (fn)	3	**Our Work [166]** / **Our Work [166]**	3	**Our Work [166]** / **Our Work [166]**
Guaranteed output delivery (god)	Impossible	[67]	3	**Our Work [166]** / **Our Work [166]**

1.4.1.2 Fast Secure Computation for 3PC and 4PC over the Internet

We present efficient constant-round constructions of 3PC and 4PC achieving strong security notions of fn and god that tolerate one active corruption. Our constructions, all based on

symmetric-key primitives are built from garbled circuits (GC). We outline our results below. For empirical purpose, the circuits of AES-128, SHA-256 and MD5 are used as benchmarks.

3PC with fairness. In the minimal network setting of pairwise-private channels, our 3PC protocol with fn consumes four rounds and involves transmission and evaluation of a *single* GC. Our protocol shows a minimal overhead of 0.06–0.16 ms, 0.03–0.8 ms, 0.21–0.5 s and 5.63–10.74 KB over the 3PC of [159] (that achieves security with selective abort), in terms of the average computation time, LAN runtime, WAN runtime and communication, where average is taken over the number of parties and the range is taken over the choice of benchmark circuits. The nominal overhead to trade fairness over abort security makes our construction a better choice for practical purposes. This protocol has a natural extension to more than 3 parties (still for one corruption) with neither inflating the round complexity nor the number of GCs.

3PC with guaranteed output delivery. With an additional broadcast channel, we present a 5-round 3PC protocols with god at the cost of communication of a *single* GC. A broadcast channel is inevitable in this regime owing to the results of [67]. We ensure that the broadcast communication is nominal and most importantly, independent of the circuit size. Our implementation, using a physical UDP broadcast channel available on LAN, shows that the average computation time, LAN runtime and communication overhead are 0.16–0.3 ms, 1.52–3 ms and 0.19–0.46 KB respectively over that of [159]. For the worst case run when the execution is stretched to 5 rounds, there is negligible change in the computation and LAN runtime, but communication overhead is witnessed to increase to a value between 0.21–0.57 KB. We do not implement the protocol in WAN as it would require an implementation of a robust broadcast protocol. When the adversary remains semi-honest, this protocol too terminates in 3 rounds and the extra communication and computation needed in the last two rounds is almost nothing.

4PC with guaranteed output delivery. In the 4-party setting, we present an efficient protocol that achieves god in five rounds, assuming just pairwise-private channels. Our protocol involves communication of a *single* GC compared to the 2-round protocol of [129] that incurs a cost of 12 GCs. Our protocol has asymmetric roles for each party involved and as a result, interestingly, our protocol gives better performance compared to the 3PC of [159]. The protocol terminates in three rounds when no malicious behaviour takes place and has minimal communication (and negligible computation) done in last two rounds. We take reading for both 3-round run and 5-round run of the protocol. For the former, our protocol shows a *gain* of 0.19–2.61 ms, 0.17–2.45 ms and 18.63–500.56 KB respectively compared to the 3PC of [159] in terms of average computation time, LAN runtime and communication. The overhead for WAN runtime is minimal and amounts to 0.02–0.31 s. When the protocol is stretched to 5 rounds,

the gains reported above remain unaffected (or witness negligible decrease). In terms of average WAN runtime, the overhead increases to 0.51 – 0.83 s, reflecting the increase in round complexity. At the expense of one extra GC, we also present a 4-round 4PC primarily as a theoretical contribution, which also terminates in three rounds when no malicious behaviour takes place.

Theoretical and Empirical Comparison. The above discussed protocols appear in [46]. We present a comparison of our protocols with the relevant state-of-the-art protocols in terms of number of GCs, rounds and security below.

Ref.	# Parties	# GCs	Rounds	Security	Broadcast
[159]	3	1	3	sa	✗
Our Work [46]	3	1	4	fn	✗
Our Work [46]	3	1	5	god	✓ [67]
[129]	4	12	2	god	✗
Our Work [46]	4	2	4	god	✗
Our Work [46]	4	1	5	god	✗

Below, we summarize the overhead or gain (indicated by **g**) of our protocols compared to the 3PC of [159] in terms of *average* computation time, LAN runtime, WAN runtime and communication cost, where the average is taken over the number of parties and the range is taken over the choice of circuits. We show in bracket the increase in the overhead or decrease in the gain for the worst case 5-round run of our 3PC and 4PC with guaranteed output delivery. With respect to our 4-round 4PC with guaranteed output delivery, in the worst case run, we save one round at the expense of one garbled circuit over our 5-round 4PC which amounts to a value in the range 72 KB – 1530 KB for the benchmark circuits.

Ref.	Computation (ms)	LAN (ms)	WAN (s)	Communication (KB)
fair 3PC	0.06 – 0.16	0.03 – 0.8	0.21 – 0.5	5.63 – 10.74
4PC with god	0.19 – 2.61 (**g**)	0.17 – 2.45 (**g**)	0.02 (+.49) – 0.31 (+.52)	18.63 (−.01) – 500.56 (−.1) (**g**)
3PC with god	0.16 – 0.3	1.52 – 3	-	0.19 (+.02) – 0.46 (+.11)

1.4.2 On the Exact Round Complexity of Best-of-both-Worlds Multiparty Computation

The two traditional streams of multiparty computation (MPC) protocols consist of– (a) protocols achieving guaranteed output delivery (god) or fairness (fn) in the honest-majority setting

[30, 56, 177, 19, 18, 72, 4] and (b) protocols achieving unanimous or selective abort (ua, sa) in the dishonest-majority setting [107, 73, 94, 44, 3, 113, 15]. The favorable presence of honest majority amongst the participants is necessary to achieve the stronger notions of god or fn[65]. With complementary challenges and techniques, each stream independently stands tall with spectacular body of work. Yet, the most worrisome shortcoming of these generic protocols is that: a protocol in *one* setting completely breaks down in the *other* setting i.e. the security promises are very rigid and specific to the setting. For example, a protocol for honest majority might no longer even be "private" or "correct" if half (or more) of the parties are corrupted. A protocol that guarantees security with ua for arbitrary corruptions cannot pull off the stronger security of god or fn even if only a "single" party is corrupt. In many real-life scenarios, it is highly unlikely for anyone to guess upfront how many parties the adversary is likely to corrupt. In such a scenario, the best a practitioner can do, is to employ the 'best' protocol from her favorite class and hope that the adversary will be within assumed corruption limit of the employed protocol. If the guess fails, the employed protocol, depending on whether it is an honest or dishonest majority protocol, will suffer from the above mentioned issues. The quest for attaining the best feasible security guarantee in the respective settings of honest and dishonest majority in a *single* protocol sets the beginning of a brand new class of MPC protocols, termed as 'Best of Both Worlds (BoBW)' [124, 134, 127]. In critical applications such as voting [138, 161], secure auctions [74], secure aggregation [42], federated learning and prediction [157, 158, 53] and many more, where privacy of the inputs of an honest party needs protection at any cost and yet a robust completion is called for (as much as theoretically feasible), BoBW protocols are arguably the best fit.

Denoting the threshold of corruption in honest and dishonest majority case by t and s respectively, an ideal BoBW MPC should promise the best possible security in each corruption scenario for any population of size n, as long as $t < n/2$ and $s < n$. Unfortunately, existing feasibility results indicate that non-reactive or standard functionalities are impossible to realise as long as $t + s \geq n$ in expected polynomial time (in the security parameter) [134, 127]. A number of meaningful relaxations were proposed in the literature to get around the impossibility of BoBW security when $t + s \geq n$ [134, 127]. The most relevant to our work is the relaxation proposed in [152] where the best possible security of god is compromised to the second-best notion of fn in the honest-majority setting.

We consider two types of BoBW MPC protocols and study their exact round complexity: (a) MPC achieving the best security of god and ua in the honest and dishonest majority setting respectively assuming $s + t < n$, referred as (god|ua)-BoBW; (b) MPC achieving second-best security notion of fn in the honest majority and the best possible security of ua in the dishonest

majority for any n, referred as (fn|ua)-BoBW. The adversary is considered malicious, rushing and polynomially-bounded in either world. The latter notion (introduced in [152]) is an elegant and meaningful relaxation that brings back the true essence of BoBW protocols with no constraint on n, apart from the natural bounds of $t < n/2$ and $s < n$. Furthermore, fn is almost as good as god for many practical applications where the adversary is rational enough and does not wish to fail the honest parties at the expense of losing its own output.

We nearly settle the exact round complexity for two classes of BoBW protocols, (god|ua)-BoBW and (fn|ua)-BoBW, under the assumption of no setup (plain model), public setup (CRS) and private setup (CRS + PKI or simply PKI). The adversary is assumed to be rushing, active and static. The parties are connected via pair-wise private channels and an additional broadcast channel. All our upper bounds are based on polynomial-time assumptions and assume black-box simulation. We summarise our results below.

(fn|ua)-BoBW. We settle the exact round complexity of this class of BoBW protocols by establishing the necessity and sufficiency of: (a) 5 rounds in the plain model and (b) 3 rounds in both the public (CRS) and private (CRS+PKI) setup setting. In the CRS model, the necessity of 3 rounds for honest-majority MPC achieving fn (and hence for (fn|ua)-BoBW) has been demonstrated in [108, 102, 166] (as discussed in Section 1.3), the former in a setting where broadcast is the only mode of communication (broadcast-only) and the latter two additionally with pairwise-private channels. However, these results do not hold in the presence of PKI. Our lower bound argument, on the other hand, is resilient to the presence of both CRS and PKI, and further holds in the presence of broadcast and pairwise-private channels.

(god|ua)-BoBW. In this regime, we demonstrate that 4, 3 and 2 are the respective lower bounds in the no-setup, public setup and private setup setting. The first lower bound follows from the fact that BoBW MPC in this class trivially subsumes the dishonest majority MPC when $t = 0$ and the lower bound for dishonest-majority MPC is 4 [95]. The last lower bound follows from the standard 2-round bound for MPC [112]. Regarding the lower bound of 3 for the public setup (CRS) setting, we point that it follows directly from the 2-round impossibility of MPC with fn for honest majority in the CRS model [108, 166, 102] for *most* values of (t, s, n) satisfying $s + t < n$. However, these existing results do not rule out the possibility of 2-round (god|ua)-BoBW MPC for $(t = 1, s > t, n \geq 4)$. (In fact the protocols of [126, 129] circumvent the 3-round lower bound for fn when $t = 1, n \geq 4$). We address this gap by giving a *unified proof* that works even for $s > t$, for all values of t (including $t = 1$). This is non-trivial and it demonstrably breaks down in the presence of PKI. The bounds are totally different from the ones for previous class, owing to the different feasibility condition of $s+t < n$. While our upper

bound falls merely one short of matching the first lower bound in case of no-setup, the upper bounds of the other two settings are tight. We leave the question of designing or alternately proving the impossibility of 4-round (god|ua)-BoBW MPC protocol as open.

The above results are currently under submission. We summarize our results along with the bounds known in the honest and dishonest majority setting below.

	No setup (Plain Model)	**Public Setup (CRS)**	**Private Setup (CRS + PKI)**
Honest Majority $t < n/2$ fn / god	Round: 3 Lower Bound: [166, 102] Upper Bound: [4, 16]	Round: 3 Lower Bound: [166, 102] Upper Bound: [108, 4, 16]	Round: 2 Lower Bound: [112] Upper Bound: [108]
Dishonest Majority $s < n$ sa / ua	Round: 4 Lower Bound: [95] Upper Bound: [113, 15, 60] (sa only)	Round: 2 Lower Bound: [112] Upper Bound: [94, 160] [92, 93, 35]	Round: 2 Lower Bound: [112] Upper Bound: [94, 160] [92, 93, 35]
(fn|ua)-**BoBW** $t < n/2, s < n$ fn & ua	Round: 5 Lower Bound: **Our Work** Upper Bound: **Our Work**	Round: 3 Lower Bound: [102, 166] Upper Bound: **Our Work**	Round: 3 Lower Bound: **Our Work** Upper Bound: **Our Work**
(god|ua)-**BoBW** $t < n/2, t + s < n$ god & ua	Round: – Lower Bound: 4 [95] Upper Bound: 5 **Our Work**	Round: 3 Lower Bound: **Our Work** Upper Bound: **Our Work**	Round: 2 Lower Bound: [112] Upper Bound: **Our Work**

1.4.3 On the Round Complexity of Fair and Robust MPC against Dynamic and Boundary Adversaries

Two of the most sought-after properties of MPC protocols are fn and god. Both these properties are trivially attainable in the presence of any number of *passive* (semi-honest) corruption where the corrupt parties follow the protocol specifications but the adversary learns the internal state of the corrupt parties. However, in the face of stringent *active* (malicious) corruption where the parties controlled by the adversary deviate arbitrarily from the protocol; fn and god can be achieved only if the adversary corrupts atmost minority of the parties (referred to as malicious minority) [65]. Opening up the possibility of corrupting parties in both passive and active style, the generalized feasibility condition for a n-party fair or robust protocol turns out to be $t_a + t_p < n$, where t_a, t_p denote the threshold for active and passive corruption, with the latter subsuming the former [122]. We emphasize that t_p is a measure of the *total* number of passive corruptions that includes the actively corrupt parties; therefore the feasibility condition $t_a + t_p < n$ implies $t_a \leq \lceil n/2 \rceil - 1$. In its most intense and diverse avatar, referred as *dynamic-admissible*, the adversary can take control of the parties in one of the ways drawn from the entire

range of admissible possibilities of (t_a, t_p) starting from $(\lceil \frac{n}{2} \rceil - 1, \lfloor n/2 \rfloor)$ to $(0, n-1)$. In a milder setting, referred as *boundary-admissible*, the adversary is restricted only to the boundary cases, namely $(\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ and $(0, n-1)$. Subsuming the traditional malicious-minority and passive-majority (majority of the parties controlled by passive adversary) setting for achieving fn and god as special cases, both dynamic as well as boundary setting give the adversary more freedom and consequently more strength to the protocols. Notably, both empower an adversary to control majority of the parties, yet ensuring the count on active corruption never goes beyond $\lceil \frac{n}{2} \rceil - 1$.

The study of protocols in dynamic and boundary setting is well motivated and driven by theoretical and practical reasons. Theoretically, the study of generalized adversarial corruptions gives deeper insight into how passive and active strategies combine to influence complexity parameters of MPC such as efficiency, security notion achieved and round complexity. Practically, the protocols in dynamic and boundary setting offer strong defence and are more tolerant and better-fit in practical scenarios where the attack can come in many unforeseen ways. Indeed, deploying such protocols in practice is far more safe than traditional malicious-minority and passive-majority protocols that completely break down in the face of boundary adversaries, let alone dynamic adversaries. For instance, consider MPC in server-aided setting where instead of assuming only actively corrupt clients and honest servers, the collusion of client-server is permitted where some of the servers can be passively monitored. This model is quite realistic as it does not contradict the reputation of the system (since the passive servers follow protocol specifications and can thereby never be exposed / caught). The option of allowing corruption in both passive and active styles is quite relevant in such scenarios.

Driven by the above credible reasons, we extend the study of exact round complexity of fair and robust (achieving god) protocols beyond the traditional malicious-minority setting [102, 108, 166] and settle the same for the regime of dynamic and boundary corruption. This is achieved via 3 lower bounds that hold assuming *both* CRS and PKI setup and 5 upper bounds that assumes CRS *alone*. In terms of network setting, while our lower bounds hold assuming *both* pairwise-private and broadcast channels, all our upper bounds use broadcast channel *alone*. All our upper bounds are generic compilers that transform a 2-round protocol achieving ua (either all honest parties obtain output or none of them do) or identifiable abort (corrupt parties are identified in case honest parties do not obtain the output) against malicious majority to a protocol achieving the stronger guarantees of fn / god against stronger adversaries (namely, dynamic and boundary adversaries). The need for CRS in our constructions stems from the underlying 2-round protocol achieving ua or identifiable abort. We leave open the question of constructing tight upper bounds or coming up with new lower bounds in the plain

model. We elaborate on the results below.

Dynamic Adversary. We recall that in this challenging setting, the adversary has the freedom to choose from the entire range of thresholds for (t_a, t_p) starting from $(\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ to $(0, n-1)$. Our first lower bound establishes that $\lceil n/2 \rceil + 1$ rounds are necessary to achieve fn against dynamic adversary. Since god is a stronger security notion, the same lower bound holds for god as well. This result not only rules out the possibility of constant-round fair protocols but also gives the *exact* lower bound. We give two matching upper bounds, one for fn and the other for god, where the former is subsumed by and acts as a stepping stone to the latter. These results completely settle the round complexity of this setting in the CRS model.

Boundary Adversary. The leap in round complexity ebb in the milder boundary adversarial setting where adversary is restricted to the boundary cases of $(\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ and $(0, n-1)$. Our two lower bounds of this setting show that 4 and 3 rounds are necessary to achieve god and fn respectively against the boundary adversary. Our first 4-round lower bound is particularly interesting, primarily due to two reasons. (1) As mentioned earlier, when n is odd, the boundary cases reduce to pure active ($t_a = t_p$ when $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$) and pure passive ($(t_a, t_p) = (0, n-1)$) corruptions. We note that security against malicious-minority and passive-majority are known to be attainable independently in just 2 rounds assuming access to CRS and PKI [108, 93, 35]. Hence, our 4-round lower bound encapsulates the difficulty in designing protocols tolerant against an adversary who can choose among his two boundary corruption types arbitrarily. (2) This lower bound can be circumvented in case of single malicious corruption i.e against a special-case boundary adversary restricted to corruption scenarios $(t_a, t_p) = (1, \lfloor n/2 \rfloor)$ and $(t_a, t_p) = (0, n-1)$. (We refer to such an adversary as special-case boundary adversary with $t_a \leq 1$). This observation augments the rich evidence in literature [172, 13, 129] which show the impact of single corruption on feasibility results. With respect to our second lower bound for fn against boundary adversary, we first note that the 3-round lower bound for fn in the presence of CRS is trivial given the feasibility results of [102, 108, 166]. However, they break down assuming access to PKI. Thus, the contribution of our second lower bound is to show that the 3-round lower bound holds for boundary adversary even in the presence of PKI. We complement these two lower bounds by three tight upper bounds. The upper bounds achieving god include a 4-round protocol for the general case and a 3-round protocol for the special-case of one malicious corruption that demonstrates the circumvention of our first lower bound. Lastly, our third upper bound is a 3-round construction achieving fn, demonstrating the tightness of our second lower bound.

The results above appeared in [169]. We summarize them in the table below with comparison

to the round complexity in the traditional settings of achieving fn and god. Since PKI (private) setup subsumes CRS (public) setup which further subsumes plain model (no setup), the lower and upper bounds are specified with their maximum tolerance and minimum need respectively amongst these setup assumptions. The results provide us further insights regarding how disparity in adversarial setting affects round complexity. Note that the round complexity of fair protocols in the CRS model against an adversary corrupting minority of parties maliciously, remains unaffected in the setting of boundary adversary; which is a stronger variant of the former. On the other hand, this switch of adversarial setting causes the lower bound of robust protocols in the model assuming both CRS and PKI to jump from 2 to 4. Lastly, the gravity of dynamic corruption on round complexity is evident in the leap from constant-rounds of $3, 4$ in the boundary corruption case to $\lceil n/2 \rceil + 1$.

Adversary	Security	Rounds	Lower bound	Upper Bound
Passive-majority	fn, god	2	[112] (private)	[93, 35] (plain)
Malicious-minority	fn, god	3	[108, 166] (public)	[4, 16] (plain)
	fn, god	2	[112] (private)	[108] (private)
Boundary	fn	**3**	**Our Work [169]** (private)	**Our Work [169]** (public)
	god	**4** (**3** when $t_a \leq 1$)	**Our Work [169]** (private)	**Our Work [169]** (public)
Dynamic	fn, god	$\lceil \frac{\mathbf{n}}{\mathbf{2}} \rceil + \mathbf{1}$	**Our Work [169]** (private)	**Our Work [169]** (public)

1.4.4 On the Power of Hybrid Networks in Multi-Party Computation

Verifiable Secret Sharing (VSS) [59, 30, 107, 69, 101] is a fundamental building block for many distributed cryptographic tasks including MPC and Byzantine Agreement [135, 1]. VSS is a two phase protocol (Sharing and Reconstruction) carried out among n parties with a designated party called *dealer* in the presence of an adversary $\mathcal{A}$ who can corrupt up to any t parties including the dealer. The goal of the VSS protocol is to let the dealer 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 (correctness), while preserving the secrecy of s until the reconstruction phase (privacy). Perfectly-secure (information-theoretically secure with no error) verifiable secret sharing (VSS) and multi-party computation (MPC) protocols among n parties secure against a coalition of t *actively* corrupt parties are known to exist if and only if

- $t < n/3$, when the underlying network is *synchronous*, and
- $t < n/4$, when the underlying network is *asynchronous*, respectively.

The above feasibility results indicate that synchrony of a network has positive impact on the fault-tolerance of distributed protocols. More generally, asynchronous protocols are known to suffer from low fault-tolerance, high communication complexity and relatively weaker guarantees compared to their synchronous counterparts. The asynchronous VSS suffers from *dealer-dependent termination* where termination of the sharing phase is guaranteed only when the dealer is honest. Similarly, asynchronous MPC suffers from *input deprivation* that refers to a property where inputs of t honest parties may be excluded from computation. All the above are supposedly caused by the following inherent and trademark difficulty in the asynchronous model.

In an asynchronous network, an honest party whose message is delayed in the network cannot be told apart from a corrupted party who did not send a message at all. So an honest party in an asynchronous protocol, unlike in a synchronous protocol, cannot wait for the messages from all the parties, as it would potentially risk him to wait infinitely. To avoid the risk, an honest party's computation in an asynchronous protocol should be carried on with the receipt of $(n - t)$ parties at any given step. Unfortunately, this may risk ignoring the values of up to t potentially honest parties at any given step. There exist well-known gaps in the feasibility results of the synchronous and asynchronous VSS and MPC that corroborate with the above inherent difficulty faced in asynchronous protocols.

We set our focus on perfectly-secure protocols and seek to close the theoretical feasibility gap of synchronous and asynchronous VSS and MPC protocols. To this effect, we explore the *hybrid* networks that is asynchronous in nature and yet supports a few synchronous rounds at the onset of a protocol execution. We wish to add and find the minimum synchrony assumption needed. More specifically, we address the following : For asynchronous protocols, we wish to bridge the fault-tolerance gap between synchronous and asynchronous protocols with minimum synchrony assumption needed, leveraging the initial synchronous rounds. For synchronous protocols, we explore if the known lower bounds on round complexity can be circumvented, leveraging the asynchronous phase available in the hybrid network. Denoting synchronous/asynchronous VSS (SVSS/AVSS) and synchronous/asynchronous MPC (SMPC/AMPC) to refer to the properties of the protocols that can be achieved in the respective networks, we present our findings below.

Results for AVSS and AMPC. Concerning AVSS and AMPC, we ask the following fundamental question: *What is the minimum number of initial synchronous rounds necessary and sufficient in a hybrid network to construct perfectly-secure AVSS and AMPC protocols with the same fault-tolerance of synchronous protocols?* On the positive side, we show that *one* synchronous round is sufficient for AVSS which is clearly optimal. On the negative side, we show

the same is not true for AMPC. Our results are summarised in the following theorems:

Theorem 1.1 *(Informal) There exists a perfectly-secure AVSS with $t < n/3$ over hybrid networks with one synchronous round.*

Theorem 1.2 *(Informal) Perfectly-secure AMPC with $n \leq 4t$ is impossible over a hybrid network that supports a single synchronous round.*

Notably *no broadcast* oracle is invoked in the synchronous round of our AVSS protocol. The latter result on AMPC implies at least *two* initial synchronous rounds are necessary for MPC. With three synchronous rounds, we design a perfectly-secure SMPC (and thus AMPC) [1] protocol. The question of designing an AMPC protocol in a hybrid network with two synchronous rounds with or without broadcast oracle access is left as an interesting open question. In this regard, we believe that our proposed AVSS protocol that achieves strong properties useful for building MPC can be an important building block. Our AVSS construction is efficient and therefore can be of independent interest too.

Results for SVSS and SMPC. We further investigate if the asynchronous phase of the hybrid network can be leveraged to save on the synchronous rounds required for SVSS and SMPC. It is known that *three* synchronous rounds are necessary and sufficient for SVSS with $t < n/3$ [101]. This makes the feasibility of SVSS with $t < n/3$ in a hybrid network with three synchronous rounds trivial. The same question seems intriguing when one or two synchronous rounds are assumed. We answer this question in the negative and prove the following theorem.

Theorem 1.3 *(Informal) Perfectly-secure SVSS with $n \leq 4t$ is impossible over a hybrid network that supports two synchronous rounds.*

In contrast, a hybrid network with *one* synchronous round is sufficient for AVSS with $t < n/3$. Since VSS is a special case of MPC, the above theorem implies the necessity of three synchronous rounds for SMPC in the same setting. We deduce the sufficiency of three synchronous rounds for SMPC over hybrid networks by combining known techniques from [63, 62, 139] and have the following theorem. In contrast, we note that one synchronous round is sufficient for *cryptographic* SMPC over hybrid networks [23].

Theorem 1.4 *(Informal) A hybrid network that supports three synchronous rounds is sufficient to achieve perfectly-secure SMPC with $t < n/3$.*

[1] SMPC realizes all properties of AMPC and provides input provision additionally

While a lower bound of three rounds is known for information-theoretic MPC over synchronous networks [102], the known popular protocols relying on 'gate-by-gate' evaluation strategy ([55, 30] and their derivatives) are shown to necessarily require a round complexity that grows linearly with the multiplicative depth of the circuit [78]. As protocols tend to run faster over asynchronous network, our SMPC over hybrid network may offer lower latency than any synchronous MPC running for rounds proportional to the circuit depth.

The above results appear in [167]. We summarize the feasibility results of SVSS/AVSS and SMPC/AMPC in hybrid networks in terms of initial synchronous rounds needed in the table below. Finding a tight upper bound for AMPC with two rounds remains an interesting open question.

Feasibility for SVSS/AVSS and SMPC/AMPC with $t < n/3$ in Hybrid networks

Security		Asynchronous	Synchronous
VSS	Necessary	One [Trivial]	Three **(Our Work)** [167]
	Sufficiency	One **(Our Work)** [167]	Three [101]
MPC	Necessary	Two **(Our Work)** [167]	Three **(Our Work)** [167]
	Sufficiency	Three **(Our Work)** [167]	Three **(Our Work)** [167]

1.5 Organization of the Thesis

We divide the thesis in three parts.

Part I comprises of two chapters (Chapters 3 - 4) that include our results related to MPC for small population that considers 3-party and 4-party setting with single active corruption (honest majority). Chapter 3 presents our results on the exact round complexity of secure three-party computation (discussed in Section 1.4.1.1). Chapter 4 presents our communication and computation efficient constant-round constructions of 3PC and 4PC achieving fn and god; suitable for high-latency networks like the Internet (discussed in Section 1.4.1.2).

Part II comprises of two chapters (Chapters 5 - 6) that extend the study of round complexity beyond the traditional settings. In Chapter 5, we overcome the demarcation of study of round complexity of MPC based on resilience (i.e honest majority or dishonest majority) and explore this question for an interesting class of protocols called the Best-of-both-Worlds MPC. This class of protocols simultaneously achieve fn / god in honest majority and ua in dishonest majority (discussed in Section 1.4.2). In Chapter 6, we overcome the demarcation of study of round complexity of MPC based on single type of corruption (i.e passive or active) and investigate the round complexity of fair and robust MPC against two powerful mixed adversaries called the dynamic and boundary adversary (discussed in Section 1.4.3).

While the above two parts include results in the computational and fully synchronous setting, the final part of the thesis involves information-theoretic setting and introduces asynchrony in the network as well. Part III comprises of one chapter (Chapter 7) that explores the power of hybrid networks to bridge the feasibility gap between perfectly-secure synchronous and asynchronous VSS and MPC protocols (discussed in Section 1.4.4).

The preliminaries and conclusion of the thesis appear in Chapter 2 and 8 respectively.

Chapter 2

Preliminaries

In this chapter, we present the relevant background including the notation, definitions, security model and an overview of some of the common primitives used in our constructions.

2.1 Notation

We denote the cryptographic security parameter by κ. A negligible function in κ is denoted by $\texttt{negl}(\kappa)$ (Definition 2.1 below). We write PPT for probabilistic polynomial-time. Composition of two functions, f and g (say, $h(x) = g(f(x))$) is denoted as $g \diamond f$. We use $[n]$ to denote the set $\{1, \ldots n\}$ and $[a, b]$ to denote the set $\{a, a+1 \ldots b\}$ when $a \leq b$ or the set $\{a, a-1, \ldots b\}$ when $a > b$. We denote by $a \leftarrow_R A$ the random sampling of a from a distribution A. For any $x \in_R \{0,1\}^m$, x^i denotes the bit of x at index i for $i \in [m]$. We use $||_{i \in [n]} x_i$ to denote concatenation of strings x_i. Let S be an infinite set and $X = \{X_s\}_{s \in S}, Y = \{Y_s\}_{s \in S}$ be distribution ensembles. We say X and Y are computationally indistinguishable, if for any PPT distinguisher and all sufficiently large $s \in S$, we have $|\Pr[(X_s) = 1] - \Pr[(Y_s) = 1]| < 1/p(|s|)$ for every polynomial $p(\cdot)$.

We use $\mathcal{P}, \mathcal{C}, \mathcal{H}$ to denote the set of all parties, set of corrupt parties and set of honest parties respectively. Lastly, as mentioned earlier, the security notions of guaranteed output delivery, fairness, unanimous abort and selective abort, identifiable abort and identifiable fairness are denoted as god, fn, ua, sa, idua and idfair respectively.

2.2 Definitions

Definition 2.1 *(Negligible functions) A function* $\texttt{negl}$ *is negligible iff* $\forall c \in \mathbb{N}\ \exists n_0 \in \mathbb{N}$ *such that* $\forall n > n_0, \texttt{negl}(n) < n^{-c}$.

Definition 2.2 *(One-Way functions) A function* $f : \{0,1\}^n \to \{0,1\}^m$ *is one-way iff* $\exists$ *poly-*

nomial time M such that $\forall x\ M(x) = f(x)$, and $\forall$ non-uniform PPT adversary $\mathcal{A}$, the following holds: $\Pr_{x \in_R \{0,1\}^n}[\mathcal{A}(f(x), 1^n) \in f^{-1}(f(x))] = \mathtt{negl}(n)$.

2.3 Security Model

We prove the security of our protocols in the standard real/ideal world paradigm. Essentially, the security of a protocol is analyzed by comparing what an adversary can do in the real execution of the protocol to what it can do in an ideal execution, that is considered secure by definition (in the presence of an incorruptible trusted party). In an ideal execution, each party sends its input to the trusted party over a perfectly secure channel, the trusted party computes the function based on these inputs and sends to each party its respective output. Informally, a protocol is secure if whatever an adversary can do in the real protocol (where no trusted party exists) can be done in the above described ideal computation. We refer to [50, 104, 146, 66] for further details regarding the security model. The security definition and the required functionalities are given below.

The "ideal" world execution involves n parties $\{P_1, P_2 \dots P_n\}$, an ideal adversary $\mathcal{S}$ who may corrupt a subset of the parties, and a functionality $\mathcal{F}$. The "real" world execution involves the PPT parties $\{P_1, P_2 \dots P_n\}$ and a real world adversary $\mathcal{A}$ who may corrupt one of the parties. We let $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(1^\kappa, z)$ denote the output pair of the honest parties and the ideal-world adversary $\mathcal{S}$ from the ideal execution with respect to the security parameter 1^κ and auxiliary input z. Similarly, let $\text{REAL}_{\Pi,\mathcal{A}}(1^\kappa, z)$ denote the output pair of the honest parties and the adversary $\mathcal{A}$ from the real execution with respect to the security parameter 1^κ and auxiliary input z.

Definition 2.3 *For $n \in \mathbb{N}$, let $\mathcal{F}$ be a functionality and let Π be a n-party protocol. We say that Π securely realizes $\mathcal{F}$ if for every PPT real world adversary $\mathcal{A}$, there exists a PPT ideal world adversary $\mathcal{S}$, corrupting the same parties, such that the following two distributions are computationally indistinguishable:*

$$\text{IDEAL}_{\mathcal{F},\mathcal{S}} \overset{c}{\approx} \text{REAL}_{\Pi,\mathcal{A}}.$$

Statistical and perfect security are defined w.r.t an unbounded adversary $\mathcal{A}$ analogously where the distributions are statistically close and identical respectively.

Target Functionalities. Taking motivation from [66, 108], we define ideal functionalities $\mathcal{F}_{\mathsf{ua}}$, $\mathcal{F}_{\mathsf{fair}}, \mathcal{F}_{\mathsf{god}}$ in Figure 2.1, Figure 2.2, Figure 2.3, Figure 2.4 for secure MPC of a function f with selective abort (sa), unanimous abort (ua), fairness (fn) and guaranteed output delivery (god)

respectively. Additionally, we also define the ideal functionalities $\mathcal{F}_{\mathsf{idua}}$ and $\mathcal{F}_{\mathsf{idfair}}$ in Figure 2.5, Figure 2.6 for identifiable abort (idua) and identifiable fairness (idfair) respectively.

Input: On message $(\mathsf{sid}, \mathsf{Input}, x_i)$ from a party P_i $(i \in [n])$, do the following: if $(\mathsf{sid}, \mathsf{Input}, *)$ message was received from P_i, then ignore. Otherwise record it internally. If x_i is outside of the domain for P_i $(i \in [n])$, consider $x_i = \texttt{abort}$.

Output to adversary: If there exists $i \in [n]$ such that $x_i = \texttt{abort}$, send $(\mathsf{sid}, \mathsf{Output}, \perp)$ to all the parties. Else, send $(\mathsf{sid}, \mathsf{Output}, y)$ to the adversary, where $y = f(x_1 \dots x_n)$.

Output to selected honest parties: Receive $(\texttt{select}, \{I\})$ from adversary, where $\{I\}$ denotes a subset of the honest parties. If an honest party belongs to I, send $(\mathsf{sid}, \mathsf{Output}, y)$, else send $(\mathsf{sid}, \mathsf{Output}, \perp)$.

Figure 2.1: Ideal Functionality for sa (selective abort)

Input: On message $(\mathsf{sid}, \mathsf{Input}, x_i)$ from a party P_i $(i \in [n])$, do the following: if $(\mathsf{sid}, \mathsf{Input}, *)$ message was received from P_i, then ignore. Otherwise record it internally. If x_i is outside of the domain for P_i $(i \in [n])$, consider $x_i = \texttt{abort}$.

Output to adversary: If there exists $i \in [n]$ such that $x_i = \texttt{abort}$, send $(\mathsf{sid}, \mathsf{Output}, \perp)$ to all the parties. Else, send $(\mathsf{sid}, \mathsf{Output}, y)$ to the adversary, where $y = f(x_1 \dots x_n)$.

Output to honest parties: Receive either `continue` or `abort` from adversary. In case of `continue`, send y to honest parties, whereas in case of `abort` send them $\perp$.

Figure 2.2: Ideal Functionality for ua (unanimous abort)

Input: On message $(\mathsf{sid}, \mathsf{Input}, x_i)$ from a party P_i $(i \in [n])$, do the following: if $(\mathsf{sid}, \mathsf{Input}, *)$ message was received from P_i, then ignore. Otherwise record it internally. If x_i is outside of the domain for P_i $(i \in [n])$, consider $x_i = \texttt{abort}$.

Output: If there exists $i \in [n]$ such that $x_i = \texttt{abort}$, send $(\mathsf{sid}, \mathsf{Output}, \perp)$ to all the parties. Else, send $(\mathsf{sid}, \mathsf{Output}, y)$ to party P_i for every $i \in [n]$, where $y = f(x_1, \dots, x_n)$.

Figure 2.3: Ideal Functionality for fn (fairness)

Input: On message $(\mathsf{sid}, \mathsf{Input}, x_i)$ from a party P_i $(i \in [n])$, do the following: if $(\mathsf{sid}, \mathsf{Input}, *)$ message was received from P_i, then ignore. Otherwise record it internally. If x_i is outside of the domain for P_i, set x_i to be some predetermined default value.

Output: Compute $y = f(x_1, \dots, x_n)$ and send $(\mathsf{sid}, \mathsf{Output}, y)$ to party P_i for every $i \in [n]$.

Figure 2.4: Ideal Functionality for god (guaranteed output delivery)

Input: On message $(\mathsf{sid}, \mathsf{Input}, x_i)$ from a party P_i $(i \in [n])$, do the following: if $(\mathsf{sid}, \mathsf{Input}, *)$ message was received from P_i, then ignore. Otherwise record it internally. If x_i is outside of the domain for P_i $(i \in [n])$, consider $x_i = (\mathtt{abort}, i)$.

Output to adversary: If there exists a set $\mathcal{I} \subset \mathcal{C}$ with $|\mathcal{I}| \geq 1$ such that $x_i = (\mathtt{abort}, i)$ for $i \in \mathcal{I}$, send $(\mathsf{sid}, \mathsf{Output}, (\perp, \mathcal{I}))$ to all the parties. Else, send $(\mathsf{sid}, \mathsf{Output}, y)$ to the adversary, where $y = f(x_1, \ldots x_n)$.

Output to honest parties: Receive either `continue` or $(\mathtt{abort}, \mathcal{I})$ from adversary where $\mathcal{I} \subset \mathcal{C}$ and $|\mathcal{I}| \geq 1$. In case of `continue`, send $(\mathsf{sid}, \mathsf{Output}, y)$ to honest parties, whereas in case of `abort` send $(\mathsf{sid}, \mathsf{Output}, (\perp, \mathcal{I}))$ to all honest parties.

Figure 2.5: Ideal Functionality for idua (identifiable abort)

Input: On message $(\mathsf{sid}, \mathsf{Input}, x_i)$ from a party P_i $(i \in [n])$, do the following: if $(\mathsf{sid}, \mathsf{Input}, *)$ message was received from P_i, then ignore. Otherwise record it internally. If x_i is outside of the domain for P_i $(i \in [n])$, consider $x_i = (\mathtt{abort}, i)$.

Output: If there exists a set $\mathcal{I} \subset \mathcal{C}$ with $|\mathcal{I}| \geq 1$ such that $x_i = (\mathtt{abort}, i)$ for $i \in \mathcal{I}$, send $(\mathsf{sid}, \mathsf{Output}, (\perp, \mathcal{I}))$ to all the parties. Else, send $(\mathsf{sid}, \mathsf{Output}, y)$ to all, where $y = f(x_1, \ldots x_n)$.

Figure 2.6: Ideal Functionality for idfair (identifiable fairness)

2.4 Primitives

2.4.1 Garbling Schemes

The term 'garbled circuit' (GC) was coined by Beaver [19], but it had largely only been a technique used in secure protocols until they were formalized as a primitive by Bellare et al. [27]. 'Garbling Schemes' as they were termed, were assigned well-defined notions of security, namely *correctness*, *privacy*, *obliviousness*, and *authenticity*. A garbling scheme $\mathcal{G}$ is characterised by a tuple of PPT algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ described below.

- $\mathsf{Gb}(1^\kappa, C)$ is invoked on a circuit C in order to produce a 'garbled circuit' C, 'input encoding information' e, and 'output decoding information' d.
- $\mathsf{En}(x, \mathsf{e})$ encodes a clear input x with encoding information e in order to produce a garbled/encoded input X.
- $\mathsf{Ev}(\mathsf{C}, \mathsf{X})$ evaluates C on X to produce a garbled/encoded output Y.
- $\mathsf{De}(\mathsf{Y}, \mathsf{d})$ translates Y into a clear output y as per decoding information d.

We give an informal intuition of the notion captured by each of the security properties, namely *correctness*, *privacy*, *obliviousness*, and *authenticity*. Correctness enforces that a correctly garbled circuit, when evaluated, outputs the correct output of the underlying circuit. Privacy aims to protect the privacy of encoded inputs. Authenticity enforces that the evaluator can only learn the output label that corresponds to the value of the function. Obliviousness captures the notion that when the decoding information is withheld, the garbled circuit evaluation leaks no information about *any* underlying clear values; be they of the input, intermediate, or output wires of the circuit. The formal definitions appear below.

Definition 2.4 *(Correctness) A garbling scheme $\mathcal{G}$ is* correct *if for all input lengths $n \leq \mathsf{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$ and inputs $x \in \{0,1\}^n$, the following probability is negligible in κ:* $\Pr\left(\mathsf{De}(\mathsf{Ev}(\mathsf{C}, \mathsf{En}(\mathsf{e}, x)), \mathsf{d}) \neq C(x) : (\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C)\right)$.

Definition 2.5 *(Static Privacy) A garbling scheme $\mathcal{G}$ is* private *if for all input lengths $n \leq \mathsf{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$, there exists a PPT simulator $\mathcal{S}_{\mathsf{prv}}$ such that for all inputs $x \in \{0,1\}^n$, for all probabilistic polynomial-time adversaries $\mathcal{A}$, the following two distributions are computationally indistinguishable:*

- (C, x) : *run* $(\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C)$*, and output* $(\mathsf{C}, \mathsf{En}(x, \mathsf{e}), \mathsf{d})$.
- $\text{IDEAL}_{\mathcal{S}_{\mathsf{prv}}}(C, C(x))$*: output* $(\mathsf{C}', \mathsf{X}, \mathsf{d}') \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, C(x))$

Definition 2.6 *(Authenticity) A garbling scheme $\mathcal{G}$ is* authentic *if for all input lengths $n \leq \mathsf{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$, inputs $x \in \{0,1\}^n$, and all PPT adversaries $\mathcal{A}$, the following probability is negligible in κ:*

$$\Pr\left(\begin{array}{c}\widehat{\mathsf{Y}} \neq \mathsf{Ev}(\mathsf{C}, \mathsf{X}) \\ \wedge \mathsf{De}(\widehat{\mathsf{Y}}, \mathsf{d}) \neq \bot\end{array} : \begin{array}{c}\mathsf{X} = \mathsf{En}(x, \mathsf{e}),\ (\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C) \\ \widehat{\mathsf{Y}} \leftarrow \mathcal{A}(\mathsf{C}, \mathsf{X})\end{array}\right).$$

Definition 2.7 *(Obliviousness) A garbling scheme $\mathcal{G}$ achieves* obliviousness *if for all input lengths $n \leq \mathsf{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$, there exists a PPT simulator $\mathcal{S}_{\mathsf{obv}}$ such that for all inputs $x \in \{0,1\}^n$, for all probabilistic polynomial-time adversaries $\mathcal{A}$, the following two distributions are computationally indistinguishable:*

- (C, x) : *run* $(\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C)$*, and output* $(\mathsf{C}, \mathsf{En}(x, \mathsf{e}))$.
- $\text{IDEAL}_{\mathcal{S}_{\mathsf{obv}}}(C)$*: output* $(\mathsf{C}', \mathsf{X}) \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C)$

We are interested in a class of garbling schemes referred to as *projective* in [27]. When garbling a circuit $C : \{0,1\}^n \to \{0,1\}^m$, a projective garbling scheme produces encoding information of the form $\mathsf{e} = (\mathsf{K}_i^0, \mathsf{K}_i^1)_{i\in[n]}$, and the encoded input X corresponding to $x = (x_i)_{i\in[n]}$ can be interpreted as $\mathsf{X} = \mathsf{En}(x, \mathsf{e}) = (\mathsf{K}_i^{x_i})_{i\in[n]}$. One of our constructions in Chapter 3 uses a *privacy-free* garbling scheme [133, 90] which demands only the properties of correctness and authenticity. Lastly, some of the constructions in Chapter 3- 4, uses an additional decoding mechanism denoted as *soft decoding* algorithm sDe [159] that can decode garbled outputs without the decoding information d. The soft-decoding algorithm must comply with correctness: $\mathsf{sDe}(\mathsf{Ev}(\mathsf{C}, \mathsf{En}(\mathsf{e}, x)), \mathsf{d}) = C(x)$ for all $(\mathsf{C}, \mathsf{e}, \mathsf{d})$. While both sDe and De can decode garbled outputs, the authenticity needs to hold only with respect to De. In practice, soft decoding in typical garbling schemes can be achieved by simply appending the truth value to each output wire label.

2.4.1.1 Adaptive Garbling

In one of our constructions in Chapter 5, we use garbling schemes with stronger privacy notion, referred to as *adaptive* [27]. Informally, such garbling schemes remain private against an adversary $\mathcal{A}$ who obtains the garbled circuit C and then selects the input x.

Definition 2.8 *(Adaptive Privacy) A garbling scheme $\mathcal{G}$ satisfies adaptive privacy if for all input lengths $n \leq \mathsf{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$, there exists a PPT simulator $\mathcal{S}_{\mathsf{ad}}$ such that for all inputs $x \in \{0,1\}^n$, for all probabilistic polynomial-time adversaries $\mathcal{A}$, the following is negligible in κ:*

$$|\Pr[\mathsf{Exp}^{\mathsf{ad}}_{\mathcal{A},\mathcal{S}_{\mathsf{ad}}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ad}}_{\mathcal{A},\mathcal{S}_{\mathsf{ad}}}(1^\lambda, 1) = 1]|$$

where the experiment $\mathsf{Exp}^{\mathsf{ad}}_{\mathcal{A},\mathcal{S}_{\mathsf{ad}}}$ is defined as follows:

- *The adversary $\mathcal{A}$ specifies the circuit C, corresponding to which it obtains (C, d) created as follows:*
  - *If $b = 0$: $(\mathsf{C}, e, d) \leftarrow \mathsf{Gb}(1^\lambda, C)$. Return (C, d)*
  - *If $b = 1$: Return $(\mathsf{C}, d) \leftarrow \mathcal{S}_{\mathsf{ad}}(1^\lambda, \theta(C), 0)$. A call with '0' indicates $\mathcal{S}_{\mathsf{ad}}$ to return (C, d) and $\theta(C)$ refers to the side-information about C. Side-information function $\theta(C)$ deterministically maps the circuit C to a string $\theta(C)$ which captures the information that the garbled circuit is allowed to reveal about C such as its size, topology (the circuit structure without the gate information), the original circuit itself or something else.*

Figure 2.7: Transformation of statically-secure garbling scheme $(\mathsf{Gb}', \mathsf{En}', \mathsf{Ev}', \mathsf{De}')$ to adaptively-secure garbling scheme $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$

$\mathsf{Gb}(1^\kappa, C)$

- $(\mathsf{C}', e', d') \leftarrow \mathsf{Gb}'(1^\kappa, C)$
- Let $C^{\mathsf{pad}} \leftarrow \{0,1\}^{|\mathsf{C}'|}$; $d^{\mathsf{pad}} \leftarrow \{0,1\}^{|d'|}$
- $\mathsf{C} \leftarrow \mathsf{C}' \oplus C^{\mathsf{pad}}, d \leftarrow d' \oplus d^{\mathsf{pad}}$
  $e \leftarrow (e', C^{\mathsf{pad}}, d^{\mathsf{pad}})$
- return (C, e, d)

$\mathsf{En}(e, x)$

- $(e', C^{\mathsf{pad}}, d^{\mathsf{pad}}) \leftarrow e$
- $\mathsf{X}' \leftarrow \mathsf{En}'(e', x)$
- Return $\mathsf{X} = (\mathsf{X}', C^{\mathsf{pad}}, d^{\mathsf{pad}})$

$\mathsf{Ev}(\mathsf{C}, \mathsf{X})$

- $(\mathsf{X}', C^{\mathsf{pad}}, d^{\mathsf{pad}}) \leftarrow \mathsf{X}$
- $\mathsf{C}' \leftarrow \mathsf{C} \oplus C^{\mathsf{pad}}$; $\mathsf{Y}' \leftarrow \mathsf{Ev}'(\mathsf{C}', \mathsf{X}')$
- Return $\mathsf{Y} = (\mathsf{Y}', d^{\mathsf{pad}})$

$\mathsf{De}(\mathsf{Y}, d)$

- $(\mathsf{Y}', d^{\mathsf{pad}}) \leftarrow \mathsf{Y}$
- $d' \leftarrow d \oplus d^{\mathsf{pad}}$
- Return $\mathsf{De}'(\mathsf{Y}', d')$

- *Next, $\mathcal{A}$ provides an input x of his choice, corresponding to it obtains the encoded input X created as follows: Return $\perp$ if x is invalid. Else,*
  - *If $b = 0$: Return $\mathsf{X} \leftarrow \mathsf{En}(e, x)$.*
  - *If $b = 1$: Let $y \leftarrow C(x)$. Return $\mathsf{X} \leftarrow \mathcal{S}_{\mathsf{ad}}(y, 1)$. A call with '1' indicates $\mathcal{S}_{\mathsf{ad}}$ to return X.*

We now recall the transformation of [26] which transforms a garbling scheme $(\mathsf{Gb}', \mathsf{En}', \mathsf{Ev}', \mathsf{De}')$ satisfying static privacy (such as Yao's garbled circuits [182]) to an adaptively-secure garbling scheme $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$. The side-information $\theta(C)$ is assumed to be the topology of the circuit C. The transformation uses one-time pads to mask C and d produced by the statically-secure scheme, and then appends the pads to X. This will ensure that the adversary learns nothing about C and d until it fully specifies function C and x.

The transformation of garbling scheme $\mathcal{G}_1 = (\mathsf{Gb}', \mathsf{En}', \mathsf{Ev}', \mathsf{De}')$ with static privacy (Definition 2.5) to garbling scheme $\mathcal{G}_2 = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ with adaptive privacy (refer Definition 2.8) is described in Figure 2.7. The idea is to use one-time pads to mask the garbled circuit C' and decoding information d' obtained by running Gb' of $\mathcal{G}_1$ and append the pads to the encoding

information and the encoded input. This ensures that the adversary learns nothing about the garbled circuit C' and decoding information d' until the input is specified. The simulator $\mathcal{S}_{\mathsf{ad}}$ of the garbling scheme $\mathcal{G}_2$, when invoked with $(1^\kappa, \theta(C), 0)$ simply returns random (C, d). In the second phase, given y, $\mathcal{S}_{\mathsf{ad}}$ runs the simulator of $\mathcal{G}_1$ (say, $\mathcal{S}$) to obtain $(\mathsf{C}', \mathsf{X}', d') \leftarrow \mathcal{S}(1^\kappa, \theta(C), y)$ and returns $\mathsf{X} = (\mathsf{X}', \mathsf{C} \oplus \mathsf{C}', d \oplus d')$. We point that while this transformation does not need $\mathcal{G}_1$ to be projective, if $\mathcal{G}_1$ is projective, so is $\mathcal{G}_2$. Thus, a projective adaptive garbled circuit can be obtained by applying this transformation on Yao's projective garbling scheme satisfying static privacy. For details, we refer to [26].

2.4.2 Non-Interactive Commitment Schemes (NICOM)

A non-interactive commitment scheme (NICOM) consists of two algorithms $(\mathsf{Com}, \mathsf{Open})$ defined as follows. Given a security parameter κ, a common parameter pp, message x and random coins r, PPT algorithm Com outputs commitment c and corresponding opening information o. Given κ, pp, a commitment and corresponding opening information (c, o), PPT algorithm Open outputs the message x. The algorithms should satisfy correctness, binding (i.e. it must be hard for an adversary to come up with two different openings of any c and *any* pp) and hiding (a commitment must not leak information about the underlying message) properties. We need this kind of strong binding as the same party who generates the pp and commitment is required to open later. Two such instantiations of NICOM based on symmetric key primitives (specifically, injective one-way functions) and the formal definitions of the properties are given below.

Properties.

- *Correctness:* For all pp, $x \in \mathcal{M}$ and $r \in \mathcal{R}$, if $(c, o) \leftarrow \mathsf{Com}(x; r)$ then $\mathsf{Open}(c, o) = x$.
- *Binding:* For all PPT adversaries $\mathcal{A}$ and all pp, it is with negligible probability that $\mathcal{A}(\mathsf{pp})$ outputs (c, o, o') such that $\mathsf{Open}(c, o) \neq \mathsf{Open}(c, o')$ and $\perp \notin \{\mathsf{Open}(c, o), \mathsf{Open}(\mathrm{c}, \mathrm{o}')\}$
- *Hiding:* For all PPT adversaries $\mathcal{A}$, the following difference is negligible (over uniform choice of pp and the random coins of $\mathcal{A}$) for all $x, x' \in \mathcal{M}$:

$$\Big| \Pr_{(\mathrm{c},\mathrm{o}) \leftarrow \mathsf{Com}(x)} [\mathcal{A}(\mathrm{c}) = 1] - \Pr_{(\mathrm{c},\mathrm{o}) \leftarrow \mathsf{Com}(x')} [\mathcal{A}(\mathrm{c}) = 1] \Big|$$

Instantiations. Here we present two instantiations of NICOM. In the random oracle model, commitment is $(\mathrm{c}, \mathrm{o}) = (\mathsf{H}(x||r), x||r) = \mathsf{Com}(x; r)$. The pp can in fact be empty. In the standard model, we can use the following bit-commitment scheme from any injective one-way function. Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation and $h : \{0,1\}^n \to \{0,1\}$ a hard core predicate for $f(\cdot)$. Then the commitment scheme for a single bit x is:

- $\mathsf{Com}(x; r)$: set $\mathrm{c} = (f(r), x \oplus h(r))$; where $r \in_R \{0,1\}^n$; set $\mathrm{o} = (r, x)$.

- $\mathsf{Open}(\mathrm{c}, \mathrm{o} = (r, x))$: return x if $\mathrm{c} = (f(r), x \oplus h(r))$; otherwise return $\perp$.

For commitment of multi-bit string, the Goldreich-Goldwasser-Micali [106] construction from a one-way permutation f can be used. Recall the GGM construction: given one-way permutation $f : \{0,1\}^k \to \{0,1\}^k$ with hard-core predicate $h : \{0,1\}^k \to \{0,1\}$, first construct a length-doubling pseudorandom generator $\mathsf{G} : \{0,1\}^k \to \{0,1\}^k$ via: $\mathsf{G}(s) = f^k(s)\ h(f^{k-1}(s)) \ldots h(s)$. Let $\mathsf{G}_0(s)$ denote the first k bits of $\mathsf{G}(s)$, and let $\mathsf{G}_1(s)$ denote the last k bits of $\mathsf{G}(s)$. For a binary string s, the commitment c can be defined as $c = \mathsf{F}(s, 0^\ell) = \mathsf{G}_0(\ldots(\mathsf{G}_0(\mathsf{G}_0(s)))\ldots)$ with $o = (s)$. It is shown in [106] that the function family $\mathcal{F} = \{F^\kappa\}$ with $F^\kappa = \{\mathsf{F}(s)\}_{s \in \{0,1\}^\kappa}$ is pseudorandom. Now, note that $\mathsf{F}(s, 0^\ell) = f^{\ell.\kappa}(s)$. Since f is a permutation, this means that the function $g(x) = \mathsf{F}(x, 0^\ell)$ is a permutation, and hence the commitment scheme has the binding property. Hiding follows from the property of PRF F [137].

2.4.2.1 Equivocal Non-interactive Commitment Schemes (eNICOM)

In some of our constructions, we also need a NICOM scheme that admits equivocation property. An equivocal non-interactive commitment (eNICOM) is a NICOM that allows equivocation of a certain commitment to any given message with the help of a trapdoor. An eNICOM comprises of the following algorithms, apart from the ones needed in NICOM:

– $\mathsf{eGen}(1^\kappa)$ returns a public parameter and a corresponding trapdoor (epp, t), where epp is used by both eCom and eOpen. The trapdoor t is used for equivocation.

– $\mathsf{Equiv}(\mathrm{c}, o', x, t)$ is invoked on a certain commitment c and its corresponding opening o', given message x and the trapdoor t and returns o such that $x \leftarrow \mathsf{eOpen}(\mathsf{epp}, \mathrm{c}, \mathrm{o})$.

An eNICOM satisfies correctness, hiding and binding properties much like the NICOM does. The hiding property of eNICOM is slightly changed compared to that of NICOM taking the equivocation property into account. This new definition implies the usual hiding definition. The formal definitions and instantiations of an eNICOM appear below.

Properties.

– *Correctness:* For all $(\mathsf{epp}, t) \leftarrow \mathsf{eGen}(1^\kappa)$, $x \in \mathcal{M}$ and $r \in \mathcal{R}$, if $(c, o) \leftarrow \mathsf{eCom}(x; r)$ then $\mathsf{eOpen}(c, o) = x$.

– *Binding:* For all $(\mathsf{epp}, t) \leftarrow \mathsf{eGen}(1^\kappa)$ and for all PPT adversaries $\mathcal{A}$, it is with negligible probability that $\mathcal{A}(\mathsf{epp})$ outputs (c, o, o') such that $\mathsf{eOpen}(c, o) \neq \mathsf{eOpen}(c, o')$ and $\perp \notin \{\mathsf{eOpen}(c, o), \mathsf{eOpen}(\mathrm{c}, \mathrm{o}')\}$

– *Hiding:* For all $(\mathsf{epp}, t) \leftarrow \mathsf{eGen}(1^\kappa)$ and for all PPT adversaries $\mathcal{A}$, and all $x, x' \in \mathcal{M}$, the following difference is negligible:

$$\Big| \Pr_{(\mathrm{c},\mathrm{o})\leftarrow\mathsf{eCom}(x)}[\mathcal{A}(\mathrm{c},\mathrm{o}) = 1] - \Pr_{(\mathrm{c},\mathrm{o})\leftarrow\mathsf{eCom}(x'),\mathrm{o}\leftarrow\mathsf{Equiv}(c,x,t)}[\mathcal{A}(\mathrm{c},\mathrm{o}) = 1] \Big|$$

Instantiations. We first present the equivocal bit commitment scheme of [71], which is based on Naor's commitment scheme [162] for single bit message. This scheme avoids the use of public-key primitives. Let $\mathsf{G} : \{0,1\}^n \to \{0,1\}^{4n}$ be a pseudorandom generator.

- $\mathsf{eGen}(1^\kappa)$: set $(\mathsf{epp}, t) = (\sigma, (r_0, r_1))$, where $\sigma = \mathsf{G}(r_0) \oplus \mathsf{G}(r_1)$
- $\mathsf{eCom}(x; r)$: set $\mathrm{c} = \mathsf{G}(r)$ if $x = 0$, else $\mathrm{c} = \mathsf{G}(r) \oplus \sigma$; set $\mathrm{o} = (r, x)$
- $\mathsf{eOpen}(\mathrm{c}, \mathrm{o} = (r, x))$: return x if $\mathrm{c} = \mathsf{G}(r) \oplus x \cdot \sigma$ (where $(\cdot)$ denotes multiplication by constant); otherwise return $\perp$.
- $\mathsf{Equiv}(\mathrm{c} = \mathsf{G}(r_0), \perp, x, t)$: return $\mathrm{o} = (r, x)$ where $r = r_0$ if $x = 0$, else $r = r_1$.

Next, we present the instantiation based on Pedersen commitment scheme [175]. Let p, q denote large primes such that q divides $(p-1)$, G_q is the unique subgroup of $\mathbb{Z}_p^*$ of order q and g is a generator of G_q.

- $\mathsf{eGen}(1^\kappa)$: set $(\mathsf{epp}, t) = ((g, h), \alpha)$ where $\alpha \in \mathbb{Z}_q$; $h = g^\alpha$
- $\mathsf{eCom}(x; r)$: set $\mathrm{c} = g^x h^r$; set $\mathrm{o} = (r, x)$.
- $\mathsf{eOpen}(\mathrm{c}, \mathrm{o} = (r, x))$: return x if $\mathrm{c} = g^x h^r$; otherwise return $\perp$.
- $\mathsf{Equiv}((\mathrm{c} = \mathsf{eCom}(x'; r'))\,, (x', r'), x, t)$: return $\mathrm{o} = (r, x)$ where $r = r' + \frac{x'-x}{t}$

While in Naor-based instantiation, a specific commitment $c = \mathsf{G}(r_0)$ can be decommitted to either 0 or 1, the Pedersen commitment scheme allows equivocation of *any* commitment.

2.4.3 Threshold Secret Sharing

Informally, a d (denoting threshold) out of n threshold secret sharing scheme distributes a secret among n participants, in such a way that any group of $d+1$ or more participants can together reconstruct the secret but no group of fewer than $d+1$ players can. Shamir secret sharing is an instance of a threshold secret-sharing [180]. We present the formal definition below.

Definition 2.9 *A d-out-of-n threshold secret sharing scheme, defined for a finite set of secrets K and a set of $\mathcal{P}$ participants, comprises of two protocols– Sharing and Reconstruction* $(\mathsf{Sh}, \mathsf{Re})$*, with the following requirements:*

- *Correctness. The secret can be reconstructed by any set of $(d+1)$ parties via* Re*. That is, $\forall s \in K$ and $\forall S = \{i_1, \ldots i_{d+1}\} \subseteq \{1, \ldots n\}$ of size $(d+1)$, $\Pr[\mathsf{Re}(s_{i_1} \ldots s_{i_{d+1}}) = s] = 1$.*

- *Privacy. Any set of d parties cannot learn anything about the secret from their shares. That is: $\forall s^1, s^2 \in K$, $\forall S = \{i_1, \ldots i_d\} \subseteq \{1, \ldots n\}$ of size d, and for every possible vector of shares $\{s_j\}_{j \in S}$, $\Pr[\{\{\mathsf{Sh}(s^1)\}_S = \{s_j\}_{i_j \in S}] = \Pr[\{\{\mathsf{Sh}(s^2)\}_S = \{s_j\}_{i_j \in S}]$, where $\{\mathsf{Sh}(s^i)\}_S$ denotes the set of shares assigned to the set S as per* Sh *when s^i is the secret for $i \in \{1, 2\}$.*

2.4.4 Symmetric-Key Encryption with Special Correctness

Definition 2.10 *A CPA-secure symmetric-key encryption scheme $\pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies special correctness if there is some negligible function ϵ such that for any message m we have:* $\Pr[\mathsf{Dec}_{k_2}(\mathsf{Enc}_{k_1}(m)) \neq \perp : k_1, k_2 \leftarrow \mathsf{Gen}(1^\kappa)] \leq \epsilon(\kappa)$

Instantiation. Here we present an instantiation borrowed from [132, 148]. Let $\mathcal{F} = \{f_k\}$ be a family of pseudorandom functions where $f_k = \{0,1\}^\kappa \to \{0,1\}^{\kappa+s}$, for $k \in \{0,1\}^\kappa$ and s is a parameter denoting message length.

- $\mathsf{Enc}_k(m) = (r, f_k(r) \oplus m0^\kappa)$ where $m \in \{0,1\}^s, r \leftarrow \{0,1\}^\kappa$ and $m0^\kappa$ denotes the concatenation of m with a string of 0s of length κ.

- $\mathsf{Dec}_k(c)$ which parses $c = (r, z)$, computes $w = z \oplus f_k(r)$ and if the last κ bits of w are 0's, it outputs the first s bits of w, else it outputs $\perp$

Part I

MPC for Small Population

Chapter 3

On the Exact Round Complexity of Secure Three-Party Computation

In this chapter, we settle the exact round complexity of three-party computation (3PC) in honest-majority setting, for a range of security notions such as selective abort (sa), unanimous abort (ua), fairness (fn) and guaranteed output delivery (god). We focus on two network settings– pairwise-private channels without and with a broadcast channel.

3.1 Introduction

The setting of 3 parties with single active corruption (honest-majority) is interesting for numerous reasons such as its relevance to practice, feasibility of attaining strong notions of fn and god as outlined in Section 1.4.1. The world of MPC for small population in honest majority setting witnesses a few more interesting phenomena. Firstly, there are evidences galore that having to handle a single corrupt party can be leveraged conveniently and taken advantage of to circumvent known lower bounds and impossibility results. A lower bound of three rounds has been proven in [102] for *fair* MPC with $t \geq 2$ and arbitrary number of parties, even in the presence of broadcast channels. [129] circumvents the lower bound by presenting a *two-round* 4PC protocol tolerating a *single* corrupt party that provides god without even requiring a broadcast channel. Verifiable secret sharing (VSS) which serves as an important tool in constructing MPC protocols are known to be impossible with $t \geq 2$ with one round in the sharing phase irrespective of the computational power of the adversary [101, 172, 13]. Interestingly enough, a perfect VSS with $(n = 5, t = 1)$ [101], statistical VSS with $(n = 4, t = 1)$ [172, 129] and cryptographic VSS with $(n = 4, t = 1)$ [13] are shown to be achievable with one round in the sharing phase. Further, assumption-wise, MPC with 3, 4 and 5 parties can be built from just One-way func-

tions (OWF) or injective one-way functions/permutations [129, 159, 52], shunning public-key primitives such as Oblivious Transfer (OT) entirely, which is the primary building block in the 2-party setting. Last but not the least, the known constructions for small population in the honest majority setting perform arguably better than the constructions with two parties while offering the same level of security. For instance, 3PC with honest majority [129, 159] allows to circumvent certain inherent challenges in malicious 2PC such as enforcing correctness of garbling which incurs additional communication. The exact round complexity is yet another measure that sets apart the protocols with three parties over the ones with two parties. For instance, 3PC protocol is achievable just in two rounds with the minimal network setting of pairwise-private channels [129]. The 2PC (and MPC with dishonest majority) protocols achieving the same level of security (with abort) necessarily require 4 rounds [136] and have to resort to a common reference string (CRS) to shoot for the best possible round complexity of 2 [112].

With the impressive list of motivations that are interesting from both the theoretical and practical viewpoint, we explore the exact round complexity of 3PC in the honest majority setting tolerating a malicious adversary. We summarize the related work in Table 3.1 (details in Section 1.3). Here NIZK [36, 82, 37] and Zaps [81] refer to the tools of non-interactive zero knowledge and 2-round witness indistinguishable proofs respectively.

Table 3.1: Relevant work in honest majority setting

Ref.	Setting	Round	Network Setting / Assumption	Security	Comments
[11]	$t < n/2$	≥ 5	private channel, Broadcast / CRS, FHE, NIZK	fn	upper bound
[108]	$t < n/2$	3	broadcast-only / CRS, FHE	god	upper bound
[108]	$t < n/2$	2	broadcast-only / CRS, PKI, FHE	god	upper bound
[16]	$t < n/2$	3	broadcast-only / Zaps, FHE	god	upper bound
[4]	$t < n/2$	3	broadcast-only / Zaps, public-key encryption	god	upper bound
[126]	$n = 5, t = 1$	2	private channel / OWF	god	upper bound
[129]	$n = 3, t = 1$	2	private channel / OWF	sa	upper bound
[129]	$n = 4, t = 1$	2	private channel / (injective) OWF	god	upper bound
[159]	$n = 3, t = 1$	3	private channel, Broadcast / PRG	ua	upper bound
[108]	$t < n/2$	3	broadcast-only / CRS	fn	lower bound
[102]	$n; t > 1$	3	private channel, Broadcast	fn	lower bound

Our Results. While details of our results appear in Section 1.4.1.1, we briefly discuss them below (for easy reference, summary appears below in Table 3.2). In the minimal setting of pairwise-private channels, 3PC with sa is known to be feasible in just two rounds, while god is infeasible to achieve irrespective of the number of rounds. Settling the quest for exact round complexity of 3PC in this setting, we show that three rounds are necessary and sufficient for ua and fn. Extending our study to the setting with an additional broadcast channel, we show that while ua is achievable in just two rounds, three rounds are necessary and sufficient for fn

and god. Our lower bound results extend for any number of parties in honest majority setting and imply tightness of several known constructions.

The fundamental concept of garbled circuits underlies all our upper bounds. Concretely, our constructions involve transmitting and evaluating only constant number of garbled circuits. Assumption-wise, our constructions rely on injective (one-to-one) one-way functions.

Table 3.2: Our results on exact round complexity of 3PC in honest majority

Security	Without Broadcast	References Necessity/Sufficiency	With Broadcast	References Necessity/Sufficiency
Selective Abort (sa)	2	[112] / [129]	2	[112] / [129]
Unanimous Abort (ua)	3	**Our Work [166] / Our Work [166]**	2	[112] / **Our Work [166]**
Fairness (fn)	3	**Our Work [166] / Our Work [166]**	3	**Our Work [166] / Our Work [166]**
Guaranteed output delivery (god)	Impossible	[67]	3	**Our Work [166] / Our Work [166]**

3.1.1 Technical Overview

Lower Bounds. We present two lower bounds– **(a)** three rounds are necessary for achieving fn in the presence of pair-wise channels and a broadcast channel; **(b)** three rounds are necessary for achieving ua in the presence of just pair-wise channels. The lower bounds are shown by taking a special 3-party function and by devising a sequence hybrid executions under different adversarial strategies, allowing to conclude any 3PC protocol computing the considered function cannot be simultaneously private and achieve fn or ua.

Upper Bounds. We present three upper bounds– **(a)** 3-round protocol with fn; **(b)** 2-round protocol with ua and **(c)** 3-round protocol with god. The former in the presence of just pairwise channels, the latter two with an additional broadcast channel. The known generic transformations such as, ua to identifiable fairness (idfair) [130] or idfair to god [66], does not help in any of our constructions. For instance, any 3-round fair protocol without broadcast cannot take the former route as it is not round-preserving and ua in two rounds necessarily requires broadcast as shown in this work. A 3-round protocol with god cannot be constructed combining both the transformations due to inflation in round complexity.

Building on the protocol of [159], the basic building block of our protocols needs two of the parties to enact the role of the garbler and the remaining party to carry out the responsibility of circuit evaluation. Constrained with just two or three rounds, our protocols are built from the parallel composition of three sub-protocols, each one with different party enacting the role of the evaluator (much like [129]). Each sub-protocol consumes two rounds. Based on the security needed, the sub-protocols deliver distinct flavours of security with 'identifiable abort'. For the fn and ua protocols, the identifiability is in the form of conflict that is local (privately known) and public/global (known to all) respectively, while for the protocol with god, it is local

identification of the corrupt. Achieving such identifiability in just two rounds (sometime without broadcast) is challenging in themselves. Pulling up the security guarantee of these subprotocols via entwining three executions to obtain the final goals of fn, ua and god constitute yet another novelty of this work.

Maintaining the input consistency across the three executions pose another challenge that are tackled via mix of novel techniques (that consume no additional cost in terms of communication) and existing tricks such as 'proof-of-cheating' or 'cheat-recovery' mechanism [145, 58]. The issue of input consistency does not appear in the construction of [159] at all, as it does not deal with parallel composition. On the other hand, the generic input consistency technique adopted in [129] can only (at the best) detect a conflict locally and cannot be extended to support the stronger form of identifiability that we need.

Below, we present the common issues faced and approach taken in all our protocols before turning towards the challenges and way-outs specific to our constructions. Two of the major efficiency bottlenecks of 2PC from garbled circuits, namely the need of multiple garbled circuits due to cut-and-choose approach and Oblivious Transfer (OT) for enabling the evaluator to receive its input in encoded form are bypassed in the 3PC scenario through two simple tricks [129, 159]. First, the garblers use common randomness to construct the same garbled circuit individually. A simple comparison of the GCs received from the two garblers allows to conclude the correctness of the GC. Since at most one party can be corrupt, if the received GCs match, then its correctness can be concluded. Second, the evaluator shares its input additively among the garblers at the onset of the protocol, reducing the problem to a secure computation of a function on the garblers' inputs alone. Specifically, assuming P_3 as the evaluator, the computation now takes inputs from P_1 and P_2 as (x_1, x_{31}) and (x_2, x_{32}) respectively to compute $C(x_1, x_2, x_{31}, x_{32}) = f(x_1, x_2, x_{31} \oplus x_{32})$. Since the garblers possess all the inputs needed for the computation, OT is no longer needed to transfer the evaluator's input in encoded form to P_3.

Next, to force the garblers to input encoding and decoding information (the keys) that are consistent with the GCs, the following technique is adopted. Notice that the issue of input consistency where a corrupt party may use different inputs as an evaluator and as a garbler in different instances of the sub-protocols is distinct and remains to be tackled separately. Together with the GC, each garbler also generates the commitment to the encoding and decoding information using the common shared randomness and communicates to the evaluator. Again a simple check on whether the set of commitments are same for both the garblers allows to conclude their correctness. Now it is infeasible for the garblers to decommit the encoded input corresponding to their own input and the evaluator's share to something that are inconsistent to the GC without being caught. Following a common trick to hide the inputs of the garblers,

the commitments on the encoding information corresponding to every bit of the garblers' input are sent in permuted order that is privy to the garblers. The commitment on the decoding information is relevant only for the fair protocol where the decoding information is withheld to force a corrupt evaluator to be fair. Namely, in the third round of the final protocol, the evaluator is given access to the decoding information only when it helps the honest parties to compute the output. This step needs us to rely on the obliviousness of our garbling scheme, apart from privacy. The commitment on the decoding information and its verification by cross-checking across the garblers are needed to prevent a corrupt party to lie later. Now we turn to the challenges specific to the constructions.

Achieving fn *in 3 rounds.* The sub-protocol for our fair construction only achieves a weak form of identifiability, a local conflict to be specific, in the absence of broadcast. Namely, the evaluator either computes the encoded output ('happy' state) or it just gets to know that the garblers are in conflict ('confused' state) in the worst case. The latter happens when it receives conflicting copies of GCs or commitments to the encoding/decoding information. In the composed protocol, a corrupt party can easily breach fairness by keeping one honest evaluator happy and the other confused in the end of round 2 and *selectively* enable the happy party to compute the output by releasing the decoding information in the third round (which was withheld until Round 2). Noting that the absence of a broadcast channel ensues conflict and confusion, we handle this using a neat trick of 'certification mechanism' that tries to enforce honest behaviour from a sender who is supposed to send a common information to its fellow participants.

A party is rewarded with a 'certificate' for enacting an honest sender and emulating a broadcast by sending the same information to the other two parties, for the common information such as GCs and commitments. This protocol internally mimics a 'Conditional Disclosure of Secrets' (CDS) protocol [103] for equality predicate, with an additional property of 'authenticity', a departure from the traditional CDS. An authenticated CDS allows the receiver to detect correct receipt of the secret/certificate (similar to authenticated encryption where the receiver knows if the received message is the desired one). As demonstrated below, the certificate allows to identify the culprit behind the confusion on one hand, and to securely transmit the decoding information from a confused honest party to the happy honest party in the third round, on the other. The certificate, being a proof of correct behaviour, when comes from an honest party, say P_i, the other honest party who sees conflict in the information distributed by P_i communicated over point-to-point channel, can readily identify the corrupt party responsible for creating the conflict in Round 3. This aids the latter party to compute the output using the

encoded output of the former honest party. The certificate further enables the latter party to release the decoding information in Round 3 in encrypted form so that the other honest party holding a certificate can decrypt it. The release of encryption is done only for the parties whose distributed information are seen in conflict, so that a corrupt party either receives its certificate or the encryption but *not* both. Consequently, it is forced to assist at least one honest party in getting the certificate and be happy to compute the output, as only a happy party releases the decoding information on clear. In a nutshell, the certification mechanism ensures that when one honest party is happy, then no matter how the corrupt party behaves in the third round, both the honest parties will compute the output in the third round. When no honest party is happy, then none can get the output. Lastly, the corrupt party must keep one honest party happy, for it to get the output.

Yet again, we use garbled circuits to implement the above where a party willing to receive a certificate acts as an evaluator for a garbled circuit implementing 'equality' check of the inputs. The other two parties act as the garblers with their inputs as the common information dealt by the evaluator. With no concern of input privacy, the circuit can be garbled in a privacy-free way [133, 90]. The certificate that is the key for output 1 is accessible to the evaluator only when it emulates a broadcast by dealing identical copies of the common information to both the other parties. Notably, [123] suggests application of garbling to realise CDS.

Achieving ua *in 2 rounds.* Moving on to our construction with ua, the foremost challenge comes from the fact that it must be resilient to any corrupt Round 2 private communication. Because there is no time to report this misbehaviour to the other honest party who may have got the output and have been treated with honest behaviour all along. Notably, in our sub-protocols, the private communication from both garblers in second round inevitably carries the encoded share of the evaluator's input (as the share themselves arrives at the garblers' end in Round 1). This is a soft spot for a corrupt garbler to selectively misbehave and cause selective abort. While the problem of transferring encoded input shares of the evaluator without relying on second round private communication seems unresolvable on the surface, our take on the problem uses a clever 'two-part release mechanism'. The first set of encoding information for random inputs picked by the garblers themselves is released in the first round privately and any misbehaviour is brought to notice in the second round. The second set of encoding information for the offsets of the random values and the actual shares of the evaluator's input is released in the second round via broadcast without hampering security, while allowing public detection. Thus the sub-protocol achieves global/public conflict and helps the final construction to exit with $\perp$ unanimously when any of the sub-protocol detects a conflict.

Achieving god *in 3 rounds.* For achieving this stronger notion, the sub-protocol here needs a stronger kind of identifiability, identifying the corrupt locally to be specific, to facilitate all parties to get output within an additional round no matter what. To this effect, our sub-protocol is enhanced so that the evaluator either successfully computes the output or identifies the corrupt party. We emphasise that the goals of the sub-protocols for ua and god, namely global conflict vs. local identification, are orthogonal and do not imply each other. The additional challenge faced in composing the executions to achieve god lies in determining the appropriate 'committed' input of the corrupt party based on which round and execution of sub-protocol it chooses to strike.

Tackling input consistency. We take a uniform approach for all our protocols. We note that a party takes three different roles across the three composed execution: an evaluator, a garbler who initiate the GC generation by picking the randomness, a co-garbler who verifies the sanity of the GC. In each instance, it gets a chance to give inputs. We take care of input consistency in two parts. First, we tie the inputs that a party can feed as an evaluator and as a garbler who initiates a GC construction via a mechanism that needs no additional communication at all. This is done by setting the permutation strings (used to permute the commitments of encoding information of the garblers) to the shares of these parties' input in a certain way. The same trick fails to work in two rounds for the case when a party acts as a garbler and a co-garbler in two different executions. We tackle this by superimposing two mirrored copies of the sub-protocol where the garblers exchange their roles. Namely, in the final sub-protocol, each garbler initiates an independent copy of garbled circuit and passes on the randomness used to the fellow garbler for verification. The previous trick is used to tie the inputs that a party feeds as an evaluator and as a garbler for the GC initiated by it (inter-execution consistency). The input consistency of a garbler for the two garbled circuits (one initiated by him and the other by the co-garbler) is taken care using 'proof-of-cheating' mechanism [145] where the evaluator can unlock the clear input of both the other parties using conflicting output wire keys (intra-execution consistency). While this works for our protocols with ua and god, the fair protocol faces additional challenges. First, based on whether a party releases a clear or encoded input, a corrupt garbler feeding two different inputs can conclude whether f leads to the same output for both his inputs, breaching privacy. This is tackled by creating the ciphertexts using conflicting input keys. Second, inspite of the above change, a corrupt garbler can launch 'selective failure attack' [156, 140] and breach privacy of his honest co-garbler. We tackle this using 'XOR-tree approach' [147] where every input bit is broken into s shares and security is guaranteed except with probability $2^{-(s-1)}$ per

input bit. We do not go for the refined version of this technique, known as probe-resistant matrix, [147, 181] for simplicity.

On the assumption needed. While the garbled circuits can be built just from OWF, the necessity of injective OWF comes from the use of commitments that need binding property for any (including adversarially-picked) public parameter. Our protocols, having 2-3 rounds, seem unable to spare rounds for generating and communicating the public parameters by a party who is different from the one opening the commitments.

On concrete efficiency. Though the focus is on the round complexity, the concrete efficiency of our protocols is comparable to Yao [182] and require transmission and evaluation of few GCs (upto 9) (in some cases we only need privacy-free GCs which permit more efficient constructions than their private counterparts [133, 90]). The broadcast communication of the optimized variants of our protocols is independent of the GC size via applying hash function. We would like to draw attention towards the new tricks such as the ones used for input consistency, getting certificate of good behaviour via garbled circuits, which may be of both theoretical and practical interest. We believe the detailed take on our protocols will help to lift them or their derivatives to practice in future.

3.1.2 Roadmap

The adversarial and network model for this work appears below. Our lower bound results appear in Section 3.2. We present our 3-round protocol with fn, 2-round protocol with ua and 3-round protocol with god in Section 3.3, 3.4 and 3.5 respectively. The respective security proofs appear in Sections 3.7.1, 3.7.2 and 3.7.3 and the common optimizations in Section 3.6. Lastly, we define authenticated CDS in Appendix 3.8 and show its realisation from one of the sub-protocol used in our 3-round fair protocol.

3.1.3 Model

We consider a set of $n = 3$ parties $\mathcal{P} = \{P_1, P_2, P_3\}$, connected by pair-wise secure and authentic channels. Each party is modelled as a probabilistic polynomial time Turing (PPT) machine. We assume that there exists a PPT adversary $\mathcal{A}$, who can actively corrupt at most $t = 1$ out of the $n = 3$ parties and make them behave in any arbitrary manner during the execution of a protocol. We assume the adversary to be static, who decides the set of t parties to be corrupted at the onset of a protocol execution. For our 2-round protocol achieving ua and 3-round protocol achieving god, a broadcast channel is assumed to exist.

3.2 Lower Bounds

In this section, we present two lower bounds– **(a)** three rounds are necessary for achieving fn in the presence of pair-wise private channels and a broadcast channel; **(b)** three rounds are necessary for achieving ua in the presence of just pair-wise private channels (and no broadcast). The second result holds even if broadcast was allowed in the first round. Our results extend for any n and t with $3t \geq n > 2t$ via standard player-partitioning technique [153]. Our results imply the following. First, sa is the best amongst the four notions (considered in this work) that we can achieve in two rounds without broadcast (from **(b)**). Second, ua as well as fn require 3 rounds in the absence of broadcast (from **(b)**). Third, broadcast does not help to improve the round complexity of fn (from **(a)**). Lastly, god requires 3 rounds with broadcast (from **(a)**). Both our lower bounds hold even in the presence of public setup (CRS model) but break down in the presence of private setup (PKI model).

3.2.1 The Impossibility of 2-round Fair 3PC

In this section, we show that it is impossible to construct a 2-round 3PC with fn for general functions. [108] presents a lower bound of three rounds assuming *non-private* point-to-point channels and a broadcast channel (their proof crucially relies on the assumption of non-private channels). [102] presents a three-round lower bound for fair MPC with $t \geq 2$ (arbitrary number of parties) in the same network setting as ours. Similar to the lower bounds of [108] and [102] (for the function of conjunction of two input bits), our lower bound result does not exploit the rushing nature of the adversary and hence holds for non-rushing adversary as well. Finally, we observe that the impossibility of 2-round 3PC for the information-theoretic setting follows from the impossibility of 2-round 3-party statistical VSS of [172] (since VSS is a special case of MPC). We now prove the impossibility formally.

Theorem 3.1 *There exist functions f such that no two-round 3PC protocol with* fn *can compute f, even in the honest majority setting and assuming access to pairwise-private and broadcast channel.*

Proof: Let $\mathcal{P} = \{P_1, P_2, P_3\}$ denote the set of 3 parties and the adversary $\mathcal{A}$ may corrupt any one of them. We prove the theorem by contradiction. We assume that there exists a two-round 3PC protocol π with fn that can compute $f(x_1, x_2, x_3)$ defined below for P_i's input x_i:

$$f(x_1, x_2, x_3) = \begin{cases} 1 & \text{if } x_2 = x_3 = 1 \\ 0 & \text{otherwise} \end{cases}$$

At a high level, we discuss two adversarial strategies $\mathcal{A}_1$ and $\mathcal{A}_2$ of $\mathcal{A}$. We consider party P_i launching $\mathcal{A}_i$ in execution Σ_i ($i \in [2]$) of π. Both the executions are assumed to be run for the same input tuple (x_1, x_2, x_3) and the same random inputs (r_1, r_2, r_3) of the three parties. (Same random inputs are considered for simplicity and without loss of generality. The same arguments hold for distribution ensembles as well.) When strategy $\mathcal{A}_1$ is launched in execution Σ_1, we would claim that by correctness of π, $\mathcal{A}$ corrupting P_1 should learn the output $y = f(x_1, x_2, x_3)$. Here, we note that the value of $f(x_1, x_2, x_3)$ depends only on the inputs of honest P_2, P_3 (i.e input values x_2, x_3) and is thus well-defined. We refer to $f(x_1, x_2, x_3)$ as the value determined by this particular combination of inputs (x_2, x_3) henceforth. Now, since $\mathcal{A}$ corrupting P_1 learnt the output, due to property of fn, P_2 should learn the output too in Σ_1. Next strategy $\mathcal{A}_2$ is designed so that P_2 in Σ_2 can obtain the same view as in Σ_1 and therefore it gets the output too. Due to fairness, we can claim that P_3 receives the output in Σ_2. A careful observation then lets us claim that P_3 can, in fact, learn the output at the end of Round 1 itself in π. Lastly, using the above observation, we show a strategy for P_3 that explicitly allows P_3 to breach privacy.

We use the following notation: Let $\mathsf{p}^r_{i\to j}$ denote the pairwise communication from P_i to P_j in round r and b^r_i denote the broadcast by P_i in round r, where $r \in [2], \{i, j\} \in [3]$. V_i denotes the view of party P_i at the end of execution of π. Below we describe the strategies $\mathcal{A}_1$ and $\mathcal{A}_2$.

$\mathcal{A}_1$: P_1 behaves honestly during Round 1 of the protocol. In Round 2, P_1 waits to receive the messages from other parties, but does not communicate at all.

$\mathcal{A}_2$: P_2 behaves honestly towards P_3 in Round 1, i.e sends the messages $\mathsf{p}^1_{2\to 3}, \mathsf{b}^1_2$ according to the protocol specification. However P_2 does not communicate to P_1 in Round 1. In Round 2, P_2 waits to receive messages from P_3, but does not communicate to the other parties.

Next we present the views of the parties in the two executions Σ_1 and Σ_2 in Table 3.3. The communications that could potentially be different from the communications in an honest execution (where all parties behave honestly) with the considered inputs and random inputs of the parties are appended with $\star$ (e.g. $\mathsf{p}^2_{1\to 3}(\star)$). We now prove a sequence of lemmas to complete our proof.

Table 3.3: Views of P_1, P_2, P_3 in Σ_1 and Σ_2

	Σ_1			Σ_2		
	V_1	V_2	V_3	V_1	V_2	V_3
Initial Input	(x_1, r_1)	(x_2, r_2)	(x_3, r_3)	(x_1, r_1)	(x_2, r_2)	(x_3, r_3)
Round 1	$\mathsf{p}^1_{2\to1}$, $\mathsf{p}^1_{3\to1}$, $\mathsf{b}^1_2, \mathsf{b}^1_3$	$\mathsf{p}^1_{1\to2}$, $\mathsf{p}^1_{3\to2}$, $\mathsf{b}^1_1, \mathsf{b}^1_3$	$\mathsf{p}^1_{1\to3}$, $\mathsf{p}^1_{2\to3}$, $\mathsf{b}^1_1, \mathsf{b}^1_2$	$-$, $\mathsf{p}^1_{3\to1}$, $\mathsf{b}^1_2, \mathsf{b}^1_3$	$\mathsf{p}^1_{1\to2}$, $\mathsf{p}^1_{3\to2}$, $\mathsf{b}^1_1, \mathsf{b}^1_3$	$\mathsf{p}^1_{1\to3}$, $\mathsf{p}^1_{2\to3}$, $\mathsf{b}^1_1, \mathsf{b}^1_2$
Round 2	$\mathsf{p}^2_{2\to1}$, $\mathsf{p}^2_{3\to1}$, $\mathsf{b}^2_2, \mathsf{b}^2_3$	$-$, $\mathsf{p}^2_{3\to2}$, b^2_3	$-$, $\mathsf{p}^2_{2\to3}$, b^2_2	$-$, $\mathsf{p}^2_{3\to1}$, b^2_3	$\mathsf{p}^2_{1\to2}(\star)$, $\mathsf{p}^2_{3\to2}$, $\mathsf{b}^2_1(\star), \mathsf{b}^2_3$	$-$, $\mathsf{p}^2_{1\to3}(\star)$, $\mathsf{b}^2_1(\star)$

Lemma 3.1 *A corrupt P_1 launching $\mathcal{A}_1$ in Σ_1 should learn the output $y = f(x_1, x_2, x_3)$.*

Proof: The proof follows easily. Since P_1 behaved honestly during Round 1, it received all the desired communication from honest P_2 and P_3 in Round 2 (refer to Table 3.3 for the view of P_1 in Σ_1 in the end of Round 2). So it follows from the correctness property that his view at the end of the protocol i.e V_1 should enable P_1 to learn the correct function output $f(x_1, x_2, x_3)$. □

Lemma 3.2 *A corrupt P_2 launching $\mathcal{A}_2$ in Σ_2 should learn the output y.*

Proof: We prove the lemma with the following two claims. First, the view of P_2 in Σ_2 subsumes the view of honest P_2 in Σ_1. Second, P_2 learns the output in Σ_1 due to the fact that the corrupt P_1 learns it and π is fair. We now prove our first claim. In Σ_1, we observe that P_2 has received communication from both P_1 and P_3 in the first round, and only from P_3 in the second round. So $\mathsf{V}_2 = \{x_2, r_2, \mathsf{p}^1_{1\to2}, \mathsf{b}^1_1, \mathsf{p}^1_{3\to2}, \mathsf{b}^1_3, \mathsf{p}^2_{3\to2}, \mathsf{b}^2_3\}$ (refer to Table 3.3). We now analyze P_2's view in Σ_2. Both P_1 and P_3 are honest and must have sent $\{\mathsf{p}^1_{1\to2}, \mathsf{b}^1_1, \mathsf{p}^1_{3\to2}, \mathsf{b}^1_3\}$ according to the protocol specifications in Round 1. Since P_3 received the expected messages from P_2 in Round 1, P_3 must have sent $\{\mathsf{p}^2_{3\to2}, \mathsf{b}^2_3\}$ in Round 2. Note that we can rule out the possibility of P_3's messages in this round having been influenced by P_1 possibly reporting P_2's misbehavior towards P_1. This holds since P_3 would send the messages in the beginning of Round 2. We do not make any assumption regarding P_1's communication to P_2 in Round 2 since P_1 has not received the expected message from P_2 in Round 1. Thus, overall, P_2's view V_2 comprises of $\{x_2, r_2, \mathsf{p}^1_{1\to2}, \mathsf{b}^1_1, \mathsf{p}^1_{3\to2}, \mathsf{b}^1_3, \mathsf{p}^2_{3\to2}, \mathsf{b}^2_3\}$ (refer to Table 3.3). Note that there may also be some additional messages from P_1 to P_2 in Round 2 which can be ignored by P_2. These are marked with '$(\star)$' in Table 3.3. A careful look shows that the view of P_2 in Σ_2 subsumes the view of honest P_2 in Σ_1. This concludes our proof. □

Lemma 3.3 *P_3 in Σ_2 should learn the output y by the end of Round 1.*

Proof: According to the previous lemma, P_2 should learn the function output in Σ_2. Due to property of fn, it must hold that an honest P_3 learns the output as well (same as obtained by P_2 i.e y with respect to x_2). First, we note that as per strategy $\mathcal{A}_2$, P_2 only communicates to P_3 in Round 1. Second, we argue that the second round communication from P_1 does not impact P_3's output computation as follows.

We observe that the function output depends only on (x_2, x_3). Clearly, Round 1 messages $\{\mathsf{p}^1_{1\to 3}, \mathsf{b}^1_1\}$ of P_1 does not depend on x_2. Next, since there is no private communication to P_1 from P_2 as per strategy $\mathcal{A}_2$, the only information that can possibly hold information on x_2 and can impact the round 2 messages of P_1 is b^1_2. However, since this is a broadcast message, P_3 holds this by the end of Round 1 itself. □

Lemma 3.4 *A corrupt P_3 violates the privacy property of π.*

Proof: The adversary corrupting P_3 participates in the protocol honestly by fixing input $x_3 = 0$. Since P_3 can get the output from P_2's and P_1's round 1 communication (Lemma 3.3), it must be true that P_3 can evaluate the function f locally by plugging in any value of x_3. (Note that P_2 and P_1's communication in round 1 are independent of the communication of P_3 in the same round.) Now a corrupt P_3 can plug in $x_3 = 1$ locally and learn x_2 (via the output $x_2 \wedge x_3$). In the ideal world, corrupt P_3 must learn nothing beyond the output 0 as it has participated in the protocol with input 0. But in the execution of π (in which P_3 participated honestly with input $x_3 = 0$), P_3 has learnt x_2. This is a clear breach of privacy as P_3 learns x_2 regardless of his input. □

Hence, we have arrived at a contradiction, completing the proof of Theorem 3.1. □

Before concluding the section, we point that the above lower bound holds even in the presence of public setup (such as the CRS model). However, it breaks down given access to private setup such as public-key infrastructure i.e PKI (as demonstrated by [170]). Essentially, the argument breaks down because Lemma 3.3 does not hold in the presence of private setup for the following reason: If a setup such as PKI is established, P_1 may hold some private information unknown to P_3 at the end of Round 1, such as the decryption of P_2's Round 1 broadcast using its exclusive secret key. This may aid in output computation by P_3; thereby it cannot be claimed that P_3 obtains the output at the end of Round 1 itself.

3.2.2 The Impossibility of 2-round 3PC with Unanimous Abort

In this section, we show that 2-round 3PC with ua is impossible to achieve in the minimal setting of pairwise-private channels.

Theorem 3.2 *There exist functions f such that no two-round 3PC protocol achieving* ua *can compute f assuming access to pairwise-private channels, even in the honest majority setting.*

Proof: We prove the theorem by contradiction. We assume that there exists a two-round 3PC protocol π achieving ua that can compute the same function $f(x_1, x_2, x_3)$ considered in the proof of Theorem 3.1.

At a high level, we discuss three adversarial strategies $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ of $\mathcal{A}$. We consider party P_1 launches $\mathcal{A}_1$ in execution Σ_1, and P_2 launches $\mathcal{A}_2, \mathcal{A}_3$ in executions Σ_2, Σ_3 of π respectively. For the sake of simplicity, the executions are assumed to be run for the same input tuple (x_1, x_2, x_3) and the same random inputs (r_1, r_2, r_3) (without loss of generality) of the three parties. We use the notation V_i^j to denote the view of party P_i at the end of execution Σ_j of π. The skeleton of the proof goes as follows: We first claim that strategy $\mathcal{A}_1$ leads to honest P_2 computing the output $y = f(x_1, x_2, x_3)$. Here, we note that the value of $f(x_1, x_2, x_3)$ depends only on the inputs of honest P_2, P_3 (i.e input values x_2, x_3) and is thus well-defined. We refer to $f(x_1, x_2, x_3)$ as the value determined by this particular combination of inputs (x_2, x_3) henceforth. Since the protocol achieves ua, honest P_3's view V_3^1 at the end of Σ_1 must lead to output computation of y by P_3. Next, strategy $\mathcal{A}_2$ executed by P_2 during Σ_2 results in P_3 having the same view as in Σ_1 i.e $\mathsf{V}_3^1 = \mathsf{V}_3^2$. Thus, honest P_3 computes the output and to preserve the property of ua, honest P_1 with view V_1^2 must also compute the output. Finally, we present a strategy $\mathcal{A}_3$ by P_2 during Σ_3 that results in P_1 having the same view as in Σ_2 i.e $\mathsf{V}_1^2 = \mathsf{V}_1^3$. It follows that honest P_1 computes the output and therefore honest P_3 with view V_3^3 must be able to compute the output too. This results in a contradiction as we conclude that if P_3's view V_3^3 enables output computation, P_3 must be able to compute the output at the end of Round 1 itself which violates privacy as proved in Lemma 3.4.

Let $\mathsf{p}_{i\to j}^r$ denote the pairwise communication from P_i to P_j in round r, where $r \in [2], \{i, j\} \in [3]$. Below we describe the strategies $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}_3$.

$\mathcal{A}_1$**:** P_1 behaves honestly during Round 1 of the protocol. In Round 2, P_1 behaves honestly towards P_2. P_1's communication to P_3 in Round 2 is according to the protocol specification for the scenario when P_1 didn't receive the expected message (or nothing) from P_2 in Round 1. In more detail, suppose $\overline{\mathsf{p}_{1\to 3}^2}$ is the message that should be sent by P_1 to P_3 according to the protocol incase P_1 didn't receive anything from P_2 in Round 1. Then as per $\mathcal{A}_1$, corrupt P_1 sends $\overline{\mathsf{p}_{1\to 3}^2}$ to P_3 in Round 2.

$\mathcal{A}_2$**:** P_2 does not communicate at all to P_1 but behaves honestly to P_3 throughout π.

$\mathcal{A}_3$**:** In Round 1, P_2 does not communicate to P_1 but behaves honestly to P_3. In Round 2, P_2 does not communicate at all.

Next we present the views of the parties in Σ_1, Σ_2 and Σ_3 in Table 3.4. Here, $\overline{\mathsf{p}^2_{1\to3}}$ is the message that should be sent by P_1 to P_3 according to the protocol incase P_1 didn't receive anything from P_2 in Round 1. Besides this, the communications that could potentially be different from the communications in an honest execution with the considered inputs and random inputs of the parties are appended with $\star$ (e.g. $\mathsf{p}^2_{1\to2}(\star)$). We now prove a sequence of lemmas to complete our proof.

Table 3.4: Views of P_1, P_2, P_3 in Σ_1, Σ_2, Σ_3

	Σ_1			Σ_2			Σ_3		
	V_1	V_2	V_3	V_1	V_2	V_3	V_1	V_2	V_3
Initial Input	(x_1, r_1)	(x_2, r_2)	(x_3, r_3)	(x_1, r_1)	(x_2, r_2)	(x_3, r_3)	(x_1, r_1)	(x_2, r_2)	(x_3, r_3)
Round 1	$\mathsf{p}^1_{2\to1}$, $\mathsf{p}^1_{3\to1}$	$\mathsf{p}^1_{1\to2}$, $\mathsf{p}^1_{3\to2}$,	$\mathsf{p}^1_{1\to3}$, $\mathsf{p}^1_{2\to3}$,	–, $\mathsf{p}^1_{3\to1}$,	$\mathsf{p}^1_{1\to2}$, $\mathsf{p}^1_{3\to2}$,	$\mathsf{p}^1_{1\to3}$, $\mathsf{p}^1_{2\to3}$,	–, $\mathsf{p}^1_{3\to1}$,	$\mathsf{p}^1_{1\to2}$, $\mathsf{p}^1_{3\to2}$,	$\mathsf{p}^1_{1\to3}$, $\mathsf{p}^1_{2\to3}$,
Round 2	$\mathsf{p}^2_{2\to1}$, $\mathsf{p}^2_{3\to1}$,	$\mathsf{p}^2_{1\to2}$, $\mathsf{p}^2_{3\to2}$,	$\overline{\mathsf{p}^2_{1\to3}}$, $\mathsf{p}^2_{2\to3}$,	–, $\mathsf{p}^2_{3\to1}$,	$\mathsf{p}^2_{1\to2}(\star)$, $\mathsf{p}^2_{3\to2}$,	$\overline{\mathsf{p}^2_{1\to3}}$, $\mathsf{p}^2_{2\to3}$,	–, $\mathsf{p}^2_{3\to1}$,	$\mathsf{p}^2_{1\to2}(\star)$, $\mathsf{p}^2_{3\to2}$,	$\overline{\mathsf{p}^2_{1\to3}}$, –

Lemma 3.5 *P_3 computes the output $y = f(x_1, x_2, x_3)$ at the end of Σ_1.*

Proof: The proof follows easily. During Σ_1, as per strategy $\mathcal{A}_1$, corrupt P_1 behaved honestly to P_2 throughout π. Therefore P_2 would compute the output $y = f(x_1, x_2, x_3)$. Due to property of ua, honest P_3 must learn the output as well. □

Lemma 3.6 *P_3 computes the output $y = f(x_1, x_2, x_3)$ at the end of Σ_2.*

Proof: We observe that the view of P_3 during Σ_1, Σ_2 is same. As per both strategies $\mathcal{A}_1$, and $\mathcal{A}_2$, P_3 receives communication from P_1, P_2 as per honest execution in Round 1. In Round 2, according to $\mathcal{A}_1$, corrupt P_1 sends $\overline{\mathsf{p}^2_{1\to3}}$ as per protocol specification for case when P_1 receives nothing from P_2 in Round 1. A similar message would be sent by honest P_1 to P_3 who did not receive anything from P_2 in Round 1 (as per $\mathcal{A}_2$) during Σ_2. It is now easy to check (refer Table 3.4) that $\mathsf{V}^1_3 = \mathsf{V}^2_3$. Finally, since V^1_3 leads to output computation of y as per Lemma 3.5, P_3's view at the end of Σ_2 i.e V^2_3 must result in P_3 computing the output y. □

Lemma 3.7 *P_3 learns the output at the end of Σ_3.*

Proof: Firstly, it follows from lemma 3.6 and property of ua that honest P_1 must compute the output at the end of Σ_2. Next, it is easy to check that $\mathsf{V}_1^2 = \mathsf{V}_1^3$ (refer Table 3.4). We can thus conclude that honest P_1 computes the output at the end of Σ_3. Therefore, honest P_3 must also be able to compute the output at the end of Σ_3 (by assumption that π achieves ua). □

Finally, we now prove that P_3 learns the output at the end of Round 1 (similar to Lemma 3.3).

Lemma 3.8 *P_3 in Σ_3 should learn the output y by the end of Round 1.*

Proof: According to lemma 3.7, P_3 should learn the function output in Σ_3. First, we note that as per strategy $\mathcal{A}_3$, corrupt P_2 only communicates to P_3 in Round 1. Second, we argue that the second round communication from P_1 does not impact P_3's output computation as follows.

We observe that the function output depends only on (x_2, x_3). Clearly, the first round messages $\{\mathsf{p}^1_{1\to 3}\}$ of P_1 does not depend on x_2. Next, since there is no communication to P_1 from P_2 as per strategy $\mathcal{A}_3$, round 2 messages of P_1 hold no information about x_2. □

If P_3 is able to compute output at the end of Round 1, we know that protocol π violates privacy (proved in Lemma 3.4). We have thus arrived at a contradiction, concluding the proof of Theorem 3.2. □

We observe that even if broadcast was allowed in the first round, all the above arguments would still hold. We state this as a corollary below.

Corollary 3.1 *There exist functions f such that no two-round 3PC protocol achieving ua can compute f assuming access to pairwise-private and broadcast channels in Round 1 and only pairwise-private channels in Round 2; even in the honest majority setting.*

Proof: We observe that the following minor tweaks to the proof of Theorem 3.2 imply Corollary 3.1: We redefine $\overline{\mathsf{p}^2_{1\to 3}}$ to be the message that should be sent by P_1 to P_3 in Round 2 according to the protocol incase P_1 didn't receive anything *privately* (over pairwise-private channel) from P_2 in Round 1 (if Round 1 includes broadcast communication from P_2, then we assume P_1 has received P_2's broadcast communication). $\mathcal{A}_1$ remains the same with $\overline{\mathsf{p}^2_{1\to 3}}$ defined as above. We emphasize that there is no broadcast channel available in Round 2 and $\overline{\mathsf{p}^2_{1\to 3}}$ is communicated via pairwise-private channel between P_1 and P_3. Strategies $\mathcal{A}_2$ and $\mathcal{A}_3$ are tweaked to include honest behavior of P_2 in broadcast communication of Round 1. It is now easy to check that the arguments of Lemma 3.5 - 3.7 hold. We can now conclude that P_3 learns the output at the end of Σ_3 where the only communication from P_2 throughout the protocol

includes broadcast communication in Round 1 and private communication to P_3 in Round 1. Finally, similar to Lemma 3.8 we can argue that P_3 learns the output at the end of Round 1 itself which violates privacy.

We clarify that while the above argument holds for the plain model and public setup (such as CRS model), it does not hold in the presence of private setup such as PKI. The argument breaks down for the same reason as demonstrated by [170] in the context of our lower bound of Section 3.2.1 (elaborated at the end of Section 3.2.1). □

Alternative functions. While it suffices to show impossibility with respect to a particular function to rule out the possibility of having generic protocols, we cite yet another function that can lead to the same conclusion. Consider a function f' that outputs the message m which is the decryption of ciphertext c (P_2's input) where the decryption key k constitutes P_3's input. All our arguments still hold except Lemma 3.4: Instead of the argument of how privacy could be breached by corrupt P_3 who gets access to output at the end of Round 1, in the context of this function f', a corrupt P_3 (who gets access to the output at the end of Round 1 itself) would be able to get decryptions of the ciphertext c corresponding to multiple keys k of his choice which violates correctness.

3.3 3-round 3PC with Fairness

This section presents a tight upper bound for 3PC achieving fn in the setting with just pair-wise private channels. Our result from Section 3.2.2 rules out the possibility of achieving fn in 2 rounds in the same setting. Our result from Section 3.2.1 further shows tightness of 3 rounds even in the presence of a broadcast channel.

Building on the intuition given in the introduction, we proceed towards more detailed discussion of our protocol. Our fair protocol is built from parallel composition of three copies of each of the following two sub-protocols: (a) Fair_i where P_i acts as the evaluator and the other two as garblers for computing the desired function f. This sub-protocol ensures that honest P_i either computes its encoded output or identifies just a conflict in the worst case. The decoding information is committed to P_i, yet not opened. It is released in Round 3 of the final composed protocol under subtle conditions as elaborated below. (b) Cert_i where P_i acts as the evaluator and the other two as garblers for computing an equality checking circuit on the common information distributed by P_i in the first round of the final protocol. Notably, though the inputs come solely from the garblers, they are originated from the evaluator and so the circuit can be garbled in a privacy-free fashion. This sub-protocol ensures either honest P_i gets its certificate, the key for output 1 (meaning the equality check passes through), or identifies a

conflict in the worst case. The second round of Cert_i is essentially an 'authenticated' CDS for equality predicate tolerating one active corruption as discussed in Appendix 3.8. Three *global* variables are maintained by each party P_i to keep tab on the conflicts and the corrupt. Namely, $\mathcal{C}_i$ to keep the identity of the corrupt, flag_j and flag_k (for distinct $i, j, k \in [3]$) as indicators of detection of conflict with respect to information distributed by P_j and P_k respectively. The sub-protocols Fair_i and Cert_i assure that if neither the two flags nor $\mathcal{C}_i$ is set, then P_i must be able to evaluate the GC successfully and get its certificate respectively.

Once $\{\mathsf{Fair}_i, \mathsf{Cert}_i\}_{i\in[3]}$ complete by the end of round 2 of the final protocol Fair, any honest party will be in one of the three states: (a) no corruption and no conflict detected ($(\mathcal{C}_i = \emptyset) \wedge (\mathsf{flag}_j = 0) \wedge (\mathsf{flag}_k = 0)$); (b) corruption detected ($\mathcal{C}_i \neq \emptyset$); (c) conflict detected $(\mathsf{flag}_j = 1) \vee (\mathsf{flag}_k = 1)$. An honest party, guaranteed to have computed its encoded output and certificate *only* in the first state, releases these as well as the decoding information for both the other parties unconditionally in the third round. In the other two states, an honest party conditionally releases only the decoding information. This step is extremely crucial for maintaining fairness. Specifically, a party that belongs to the second state, releases the decoding information only to the party identified to be honest. A party that belongs to the third state, releases the decoding information in encrypted form *only* to the party whose distributed information are not agreed upon, so that the encryption can be unlocked only via a valid certificate. A corrupt party will either have its certificate or the encrypted decoding information, but *not* both. The former when it distributes its common information correctly and the latter when it does not. The only way a corrupt party can get its decoding information is by keeping one honest party in the first state, in which case both the honest parties will be able to compute the output as follows. The honest party in state one, say P_i, either gets it decoding information on clear or in encrypted form. The former when the other honest party, P_j is in the first or second state and the latter when P_j is in the third state. P_i retrieves the decoding information no matter what, as it also holds the certificate to open the encryption. An honest party P_j in the second state, on identifying P_i as honest, takes the encoded output of P_i and uses its own decoding information to compute the output. The case for an honest party P_j in the third state is the most interesting. Since honest P_i belongs to the first state, a corrupt party must have distributed its common information correctly as otherwise P_i will find a conflict and would be in third state. Therefore, P_j in the third state must have found P_i's information on disagreement due the corrupt party's misbehaviour. Now, P_i's certificate that proves his correct behaviour, allows P_j to identify the corrupt, enter into the second state and compute the output by taking the encoded output of honest P_i. In the following, we describe execution Fair_i assuming input consistency, followed by Cert_i. Entwining the six executions, tackling the input consistency and

the final presentation of protocol Fair appear in the end.

3.3.1 Protocol Fair_i

At a high level, Fair_i works as follows. In the first round, the evaluator shares its input additively between the two garblers making the garblers the sole input contributors to the computation. In parallel, each garbler initiates construction of a GC and commitments on the encoding and decoding information. While the GC and the commitments are given to the evaluator P_i, the co-garbler, acting as a verifier, additionally receives the source of the used randomness for GC and openings of commitments. Upon verification, the co-garbler either approves or rejects the GC and commitments. In the former case, it also releases its own encoded input and encoded input for the share of P_i via opening the commitments to encoding information in second round. In the latter case, P_i sets the flag corresponding to the generator of the GC to true. Failure to open a verified commitment readily exposes the corrupt to the evaluator. If all goes well, P_i evaluates both circuits and obtains encoded outputs. The correctness of the evaluated GC follows from the fact that it is either constructed or scrutinised by a honest garbler. The decoding information remains hidden (yet committed) with P_i and the obliviousness of GC ensures that P_i cannot compute the output until it receives the correct opening.

To avoid issues of adaptivity, the GCs are not sent on clear in the first round to P_i who may choose its input based on the GCs. Rather, a garbler sends a commitment to its GC to P_i and it is opened only by the co-garbler after successful scrutiny. The correctness of evaluated GC still carries over as a corrupt garbler cannot open to a different circuit than the one committed by an honest garbler by virtue of the binding property of the commitment scheme. We use an eNICOM for committing the GCs and decoding information as equivocation is needed to tackle a technicality in the security proof. The simulator of our final protocol needs to send the commitments on GC, encoding and decoding information without having access to the input of an evaluator P_i (and thus also the output), while acting on behalf of the honest garblers in Fair_i. The eNICOM cannot be used for the encoding information, as they are opened by the ones who generate the commitments and eNICOM does not provide binding in such a case. Instead, the GCs and the decoding information are equivocated based on the input of the evaluator and the output.

Protocol Fair_i appears in Figure 3.1 where P_i returns encoded outputs $\mathsf{Y}_i = (\mathsf{Y}_i^j, \mathsf{Y}_i^k)$ (initially set to $\perp$) for the circuits created by P_j, P_k, the commitments to the respective decoding information $C_j^{\mathsf{dec}}, C_k^{\mathsf{dec}}$ and the flags $\mathsf{flag}_j, \mathsf{flag}_k$ (initially set to false) to be used in the final protocol. The garblers output their respective corrupt set, flag for the fellow garbler and opening for the decoding information corresponding to its co-garbler's GC and *not* its own. This

is to ensure that it cannot break the binding of eNICOM which may not necessarily hold for adversarially-picked public parameter.

Protocol Fair_i

Inputs: Party P_α has x_α for $\alpha \in [3]$.

Common Inputs: The circuit $C(x_1, x_2, x_3, x_4)$ that computes $f(x_1, x_2, x_3 \oplus x_4)$.

Output: A garbler P_l ($l \in \{j,k\}$) outputs corrupt set $\mathcal{C}_l, \mathsf{flag}_{\{j,k\}\setminus l}$ and O_i^{dec}. P_i outputs ($\mathcal{C}_i$, $\mathsf{Y}_i = (\mathsf{Y}_i^j, \mathsf{Y}_i^k), C_j^{\mathsf{dec}}, C_k^{\mathsf{dec}}, \mathsf{flag}_j, \mathsf{flag}_k$) where Y_i denote a pair of encoded outputs or $\perp$.

Primitives: A garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ that is correct, private and oblivious, a NICOM $(\mathsf{Com}, \mathsf{Open})$, an eNICOM $(\mathsf{eGen}, \mathsf{eCom}, \mathsf{eOpen}, \mathsf{Equiv})$ and a PRG G.

Round 1:

– P_i randomly secret shares his input x_i as $x_i = x_{ij} \oplus x_{ik}$ and sends x_{ij} to P_j and x_{ik} to P_k.

– P_l for $l \in \{j,k\}$ samples $s_l \in_R \{0,1\}^\kappa$, epp_l and pp_l for G, eNICOM and NICOM resp. and:

  ◦ compute garbled circuit $(\mathsf{C}_l, \mathsf{e}_l, \mathsf{d}_l) \leftarrow \mathsf{Gb}(1^\kappa, C)$ using randomness from $\mathsf{G}(s_l)$. Assume $\{\mathsf{K}^0_{l\alpha}, \mathsf{K}^1_{l\alpha}\}_{\alpha\in[\ell]}$, $\{\mathsf{K}^0_{l(\ell+\alpha)}, \mathsf{K}^1_{l(\ell+\alpha)}\}_{\alpha\in[\ell]}$, $\{\mathsf{K}^0_{l(2\ell+\alpha)}, \mathsf{K}^1_{l(2\ell+\alpha)}\}_{\alpha\in[2\ell]}$ denote the encoding information for the input of P_j, P_k and the secret shares of P_i respectively.

  ◦ compute commitments for GC and decoding information. $(\mathrm{c}_l, \mathrm{o}_l) \leftarrow \mathsf{eCom}(\mathsf{epp}_l, \mathsf{C}_l)$ and $(\mathrm{c}_l^{\mathsf{dec}}, \mathrm{o}_l^{\mathsf{dec}}) \leftarrow \mathsf{eCom}(\mathsf{epp}_l, \mathsf{d}_l)$.

  ◦ sample permutation strings $p_{lj}, p_{lk} \in_R \{0,1\}^\ell$ for the inputs of P_j and P_k. Compute commitments to encoding information as: for $b \in \{0,1\}$, $(\mathrm{c}^b_{l\alpha}, \mathrm{o}^b_{l\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}_l, \mathsf{e}^{p^\alpha_{lj}\oplus b}_{l\alpha})$, $(\mathrm{c}^b_{l(\ell+\alpha)}, \mathrm{o}^b_{l(\ell+\alpha)}) \leftarrow \mathsf{Com}(\mathsf{pp}_l, \mathsf{e}^{p^\alpha_{lk}\oplus b}_{l(\ell+\alpha)})$ when $\alpha \in [\ell]$, $(\mathrm{c}^b_{l(2\ell+\alpha)}, \mathrm{o}^b_{l(2\ell+\alpha)}) \leftarrow \mathsf{Com}(\mathsf{pp}_l, \mathsf{e}^b_{l(2\ell+\alpha)})$ when $\alpha \in [2\ell]$.

  ◦ send $\mathcal{D}_l = (\mathsf{epp}_l, \mathsf{pp}_l, \mathrm{c}_l, \{\mathrm{c}^b_{l\alpha}, \}_{\alpha\in[4\ell], b\in\{0,1\}}, \mathrm{c}_l^{\mathsf{dec}})$ to both the other parties and send $\{s_l, p_{lj}, p_{lk}, \mathrm{o}_l, \{\mathrm{o}^b_{l\alpha}, \}_{\alpha\in[4\ell], b\in\{0,1\}}, \mathrm{o}_l^{\mathsf{dec}}\}$ only to co-garbler $P_{\{j,k\}\setminus l}$.

– P_j sets $\mathcal{C}_j = P_k$ if $\mathcal{D}_k$ and $\{s_k, p_{kj}, p_{kk}, \mathrm{o}_k, \{\mathrm{o}^b_{k\alpha}, \}_{\alpha\in[4\ell], b\in\{0,1\}}, \mathrm{o}_k^{\mathsf{dec}}\}$ are inconsistent. Else, set $O_i^{\mathsf{dec}} = \mathrm{o}_k^{\mathsf{dec}}$.

  P_k performs similar steps for the values received from P_j.

Round 2:

– P_i sends $\mathcal{D}_j$ to P_k and $\mathcal{D}_k$ to P_j. P_j sets $\mathsf{flag}_k = 1$ if $\mathcal{D}_k$ received from P_i and P_k does not match. Similar step is executed by P_k.

– P_j computes the indicator strings $m_{jj} = p_{jj} \oplus x_j, m_{kj} = p_{kj} \oplus x_j$ for its inputs. If $P_k \notin \mathcal{C}_j$, then send $\left(\mathtt{OK}, \mathcal{D}_k, (\mathrm{o}_k, \{\mathrm{o}_{k\alpha}^{m_{kj}^\alpha}, \mathrm{o}_{k(2\ell+\alpha)}^{x_{ij}^\alpha}\}_{\alpha\in[\ell]}, m_{kj}), (\{\mathrm{o}_{j\alpha}^{m_{jj}^\alpha}, \mathrm{o}_{j(2\ell+\alpha)}^{x_{ij}^\alpha}\}_{\alpha\in[\ell]}, m_{jj})\right)$ to P_i. Else, send $\mathtt{nOK}$ to P_i. P_k performs similar steps.

– (Local Computation) P_i sets $\mathsf{Y}_i^j = \perp$ and $\mathsf{flag}_j = 1$ when **(a)** P_k sent $\mathtt{nOK}$ or **(b)** $\mathcal{D}_j$ sent by P_j and P_k do not match. Otherwise, P_i sets $C_j^{\mathsf{dec}} = c_j^{\mathsf{dec}} \in \mathcal{D}_j$ and does:

  ◦ open $\mathsf{C}_j \leftarrow \mathsf{eOpen}(\mathsf{epp}_j, \mathrm{c}_j, \mathrm{o}_j)$ with o_j received from P_k. Set $\mathcal{C}_i = P_k$ if $\mathsf{C}_j = \perp$.

  ◦ open $\mathsf{X}_j^\alpha = \mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{j\alpha}^{m_{jj}^\alpha}, \mathrm{o}_{j\alpha}^{m_{jj}^\alpha})$, $\mathsf{X}_{ij}^\alpha = \mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{j(2\ell+\alpha)}^{x_{ij}^\alpha}, \mathrm{o}_{j(2\ell+\alpha)}^{x_{ij}^\alpha})$, for $\alpha \in [\ell]$, for the opening received from P_j and the commitments taken from $\mathcal{D}_j$. Include P_j in $\mathcal{C}_i$ if any of the opened input labels above is opened to $\perp$.

  ◦ open $\mathsf{X}_k^\alpha = \mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{j(\ell+\alpha)}^{m_{jk}^\alpha}, \mathrm{o}_{j(\ell+\alpha)}^{m_{jk}^\alpha})$ and $\mathsf{X}_{ik}^\alpha = \mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{j(3\ell+\alpha)}^{x_{ik}^\alpha}, \mathrm{o}_{j(3\ell+\alpha)}^{x_{ik}^\alpha})$ for $\alpha \in [\ell]$, for the opening received from P_k and the commitments taken from $\mathcal{D}_j$. Include P_k in $\mathcal{C}_i$ if any of the opened input labels above is opened to $\perp$.

  ◦ If $\mathcal{C}_i = \emptyset$, set $\mathsf{X} = \mathsf{X}_j|\mathsf{X}_k|\mathsf{X}_{ij}|\mathsf{X}_{ik}$, run $\mathsf{Y}_i^j \leftarrow \mathsf{Ev}(\mathsf{C}_j, \mathsf{X})$. Else set $\mathsf{Y}_i^j = \perp$

  Similar steps for C_k will be executed to compute Y_i^k, populate $\mathcal{C}_i$ and update flag_k.

Figure 3.1: Protocol Fair_i

Lemma 3.9 *During* Fair_i, $P_\beta \notin \mathcal{C}_\alpha$ *holds for honest* P_α, P_β.

Proof: An honest P_α would include P_β in $\mathcal{C}_\alpha$ only if one of the following hold: (a) Both are garblers and P_β sends commitments to garbled circuit, encoding and decoding information inconsistent with the randomness and openings shared privately with P_α (b) P_α is an evaluator and P_β is a garbler and either (i) P_β's opening of a committed garbled circuit fails or (ii) P_β's opening of a committed encoded input fails. It is straightforward to verify that the cases will never occur for honest (P_α, P_β). □

Lemma 3.10 *If honest* P_i *has* $\mathcal{C}_i = \emptyset$ *and* $\mathsf{flag}_j = \mathsf{flag}_k = 0$, *then* $\mathsf{Y}_i = (\mathsf{Y}_i^j, \mathsf{Y}_i^k) \neq \perp$.

Proof: According to Fair_i, P_i fails to compute Y_i when it identifies the corrupt or finds a mismatch in the common information $\mathcal{D}_j$ or $\mathcal{D}_k$ or receives a $\mathtt{nOK}$ signal from one of its garblers. The first condition implies $\mathcal{C}_i \neq \emptyset$. The second condition implies, P_i would have set either flag_j or flag_k to true. For the third condition, if P_j sends $\mathtt{nOK}$ then P_i would set $\mathsf{flag}_k = 1$. Lastly, if P_k sends $\mathtt{nOK}$, then P_i sets $\mathsf{flag}_j = 1$. Clearly when $\mathcal{C}_i = \emptyset \wedge \mathsf{flag}_j = 0 \wedge \mathsf{flag}_k = 0$, P_i evaluates both $\mathsf{C}_j, \mathsf{C}_k$ and obtains $\mathsf{Y}_i = (\mathsf{Y}_i^j, \mathsf{Y}_i^k) \neq \perp$. □

3.3.2 Protocol Cert_i

When a party P_i in Fair_i is left in a confused state and has no clue about the corrupt, it is in dilemma on whether or whose encoded output should be used to compute output and who should it release the decoding information (that it holds as a garbler) to in the final protocol. Protocol Cert_i, in a nutshell, is introduced to help a confused party to identify the corrupt and take the honest party's encoded output for output computation, on one hand, and to selectively deliver the decoding information only to the other honest party, on the other. Protocol Cert_i implements evaluation of an equality checking function that takes inputs from the two garblers and outputs 1 when the test passes and outputs the inputs themselves otherwise. In the final protocol, the inputs are the common information (GCs and commitments) distributed by P_i across all executions of Fair_j. The certificate is the output key corresponding to output 1. Since input privacy is not a concern here, the circuit is enough to be garbled in privacy-free way and authenticity of garbling will ensure a corrupt P_i does not get the certificate. Cert_i follows the footstep of Fair_i with the following simplifications: (a) Input consistency need not be taken care across the executions implying that it is enough one garbler alone initiates a GC and the other garbler simply extends its support for verification. To divide the load fairly, we assign garbler P_j where $i = (j+1) \mod 3$ to act as the generator of GC in Cert_i. (b) The decoding information need not be committed or withheld. We use soft decoding that allows immediate decoding.

Similar to Fair_i, at the end of the protocol, either P_i gets its certificate (either the key for 1 or the inputs themselves), or sets its flags (when GC and commitment do not match) or sets its corrupt set (when opening of encoded inputs fail). P_i outputs its certificate, the flag for the GC generator and corrupt set, to be used in the final protocol. The garblers output the key for 1, flag for its fellow garbler and the corrupt set. Notice that, when Cert_i is composed in the bigger protocol, P_i will be in a position to identify the corrupt when the equality fails and the certificate is the inputs fed by the garblers. The protocol appears in Figure 3.2.

Protocol Cert_i

Common Inputs: The circuit $C(\gamma_j, \gamma_k)$ that outputs 1 if $(\gamma_j = \gamma_k)$ and $(0, \gamma_j, \gamma_k)$ otherwise. For distinct $i, j, k \in [3]$, P_i is assumed to be the evaluator and (P_j, P_k) as the garblers. We assume $i = (j+1) \mod 3, k = (j+2) \mod 3$.

Primitives: A correct, authentic, privacy-free garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ that has the property of *soft decoding*, a PRG G, a NICOM $(\mathsf{Com}, \mathsf{Open})$

Output: A garbler P_l for $l \in \{j, k\}$ outputs corrupt set $\mathcal{C}_l$ and pad_i. P_i outputs

$(\mathbf{cert}_i, \mathcal{C}_i, \mathsf{flag}_j, \mathsf{flag}_k)$. Garbler P_k additionally outputs flag_j.

Round 1: P_j does the following:

- Choose a seed $s_i \in_R \{0,1\}^\kappa$ for G and construct a garbled circuit $(\mathsf{C}_i, \mathsf{e}_i, \mathsf{d}_i) \leftarrow \mathsf{Gb}(1^\kappa, C)$. Generate commitment on garbled circuit C_i as $(\mathsf{c}_i, \mathsf{o}_i) \leftarrow \mathsf{Com}(\mathsf{C}_i)$ and on the encoding information e_i as $(\mathrm{c}_i, \mathrm{o}_i) \leftarrow \mathsf{Com}(\mathsf{e}_i)$ using randomness from $\mathsf{G}(s_i)$. Let $\mathcal{W}_i = \{\mathsf{c}_i, \mathrm{c}_i\}$. Send $(s_i, \mathcal{W}_i)$ to P_k and $\mathcal{W}_i$ to P_i.
- (Local Computation by P_k) P_k adds P_j to $\mathcal{C}_k$ if $(s_i, \mathcal{W}_i)$ are inconsistent and is not as per what an honest P_j should do. P_j and P_k output pad_i equals to the key for output 1 of C_i.

Round 2:

- P_i sends $\mathcal{W}_i$ to P_k. P_k sets $\mathsf{flag}_j = 1$ if $\mathcal{W}_i$ received from P_i and P_j is not identical.
- P_j opens its encoded input X (corresponding to γ_j) to P_i by sending the opening of the corresponding commitment in c_i.
- If $P_j \in \mathcal{C}_k$, P_k sends $\mathtt{nOK}$ to P_i. Else P_k sends $\mathcal{W}_i$, opening for garbled circuit o_i and its encoded input X_k (for γ_k) to P_i.
- (Local Computation by P_i) If P_i does not receive identical $\mathcal{W}_i$ from P_j and P_k or receives $\mathtt{nOK}$ from P_k, P_i sets $\mathbf{cert}_i = \perp$ and $\mathsf{flag}_j = 1$. Else, P_i uses the opening information sent by P_j, P_k to retrieve $\mathsf{X}_j, \mathsf{X}_k$. P_i adds P_l ($l \in \{j,k\}$) to $\mathcal{C}_i$ and sets $\mathbf{cert}_i = \perp$ if any of the openings sent by P_l result in $\perp$. Else, P_i runs $\mathsf{Y} \leftarrow \mathsf{Ev}(\mathsf{C}_i, \mathsf{X}_j, \mathsf{X}_k)$. If $\mathsf{sDe}(\mathsf{Y}) = 1$, then set $\mathbf{cert}_i = \mathsf{Y}$, else set $\mathbf{cert}_i = (\gamma'_j, \gamma'_k)$ where these two are decoded from Y.

Figure 3.2: Protocol Cert_i

Lemma 3.11 *During* Cert_i, $P_\beta \notin \mathcal{C}_\alpha$ *holds for honest* P_α, P_β.

Proof: An honest P_α would include P_β in $\mathcal{C}_\alpha$ only if one of the following holds: (a) P_β sends inconsistent $(s_\beta, \mathcal{W}_\beta)$ to P_α. (b) P_β's opening of committed encoded input or garbled circuit fails. It is straightforward to verify that the cases will never occur for honest (P_β, P_α). □

Lemma 3.12 *If an honest* P_i *has* $\mathcal{C}_i = \emptyset$ *and* $\mathsf{flag}_j = \mathsf{flag}_k = 0$, *then,* $\mathbf{cert}_i \neq \perp$.

Proof: The proof follows easily from the steps of the protocol. □

3.3.3 Protocol Fair

Building on the intuition laid out before, we only discuss input consistency that is taken care in two steps: Inter-input consistency (across executions) and intra-input consistency (within an

execution). In the former, P_i's input as an evaluator in Fair_i is tied with its input committed as garblers for its own garbled circuits in Fair_j and Fair_k. In the latter, the consistency of P_i's input for both garbled circuits in Fair_j (and similarly in Fair_k) is tackled. We discuss them one by one.

We tackle the former in a simple yet clever way without incurring any additional overhead. We explain the technique for enforcing P_1's input consistency on input x_1 as an evaluator during Fair_1 and as a garbler during $\mathsf{Fair}_2, \mathsf{Fair}_3$ with respect to his GC C_1. Since the protocol is symmetric in terms of the roles of the parties, similar tricks are adopted for P_2 and P_3. Let in the first round of Fair_1, P_1 shares its input x_1 by handing x_{12} and x_{13} to P_2 and P_3 respectively. Now corresponding to C_1 during Fair_2, P_1 and P_3 who act as the garblers use x_{13} as the permutation vector p_{11} that defines the order of the commitments of the bits of x_1. Now input consistency of P_1's input is guaranteed if m_{11} transferred by P_1 in Fair_2 is same as x_{12}, P_1's share for P_2 in Fair_1. For an honest P_1, the above will be true since $m_{11} = p_{11} \oplus x_1 = x_{13} \oplus x_1 = x_{12}$. If the check fails, then P_2 identifies P_1 as corrupt. This simple check forces P_1 to use the same input in both Fair_1 and Fair_2 (corresponding to C_1). A similar trick is used to ensure input consistency of the input of P_1 across Fair_1 and Fair_3 (corresponding to C_1) where P_1 and P_2 who act as the garblers use x_{12} as the permutation vector p_{11} for the commitments of the bits of x_1. The evaluator P_3 in Fair_3 checks if m_{11} transferred by P_1 in Fair_3 is same as x_{13} that P_3 receives from P_1 in Fair_1. While the above technique enforces the consistency with respect to P_1's GC, unfortunately, the same technique cannot be used to enforce P_1's input consistency with respect to C_2 in Fair_3 (or Fair_2) since p_{21} cannot be set to x_{12} which is available to P_2 only at the end of first round. While, P_2 needs to prepare and broadcast the commitments to the encoding information in jumbled order as per permutation string p_{21} in the first round itself. We handle it differently as below.

The consistency of P_i's input for both garbled circuits in Fair_j (and similarly in Fair_k) is tackled via 'cheat-recovery mechanism' [145]. We explain with respect to P_1's input in Fair_3. P_2 prepares a ciphertext (cheat recovery box) with the input keys of P_1 corresponding to the mismatched input bit in the two garbled circuits, C_1 and C_2 in Fair_3. This ciphertext encrypts the the input shares of garblers that P_3 misses, namely, x_{12} and x_{21}. This would allow P_3 to compute the function on clear inputs directly. To ensure that the recovered missing shares are as distributed in Fair_1 and Fair_2, the shares are not simply distributed but are committed via NICOM by the input owners and the openings are encrypted by the holders. Since there is no way for an evaluator to detect any mismatch in the inputs to and outputs from the two GCs as they are in encoded form, we use encryption scheme with special correctness (Definition 2.10) to enable the evaluator to identify the relevant decryptions. Crucially, we depart from

the usual way of creating the cheat recovery boxes using conflicting encoded outputs. Based on whether the clear or encoded output comes out of honest P_3 in round 3, corrupt garbler P_1 feeding two different inputs to C_1 and C_2 can conclude whether its two different inputs lead to the same output or not, breaching privacy. Note that the decoding information cannot be given via this cheat recovery box that uses conflicting encoded outputs as key, as that would result in circularity.

Despite using the above fix, the mechanism as discussed above is susceptible to 'selective failure attack', an attack well-known in the 2-party domain. While in the latter domain, the attack is launched to breach the privacy of the evaluator's input based on whether it aborts or not. Here, a corrupt garbler can prepare the ciphertexts in an incorrect way and can breach privacy of its honest co-garbler based on whether clear or encoded output comes out of the evaluator. We elaborate the attack in Fair_3 considering a corrupt P_1 and single bit inputs. P_1 is supposed to prepare two ciphertexts corresponding to P_2's input bit using the following key combinations– (a) key for 0 in C_1 and 1 in C_2 and (b) vice-versa. Corrupt P_1 may replace one of the ciphertexts using key based on encoded input 0 of P_2 in both the GCs. In case P_2 indeed has input 0 (that he would use consistently across the 2 GCs during Fair_3), then P_3 would be able to decrypt the ciphertext and would send clear output in Round 3. P_1 can readily conclude that P_2's input is 0. This attack is taken care via the usual technique of breaking each input bit to s number of xor-shares, referred as 'XOR-tree approach' [147] (probe-resistance matrix [147, 181] can also be used; we avoid it for simplicity). The security is achieved except with probability $2^{-(s-1)}$.

Given that input consistency is enforced, at the end of round 2, apart from the three states– (a) no corruption and no conflict detected (b) corrupt identified (c) conflict detected, a party can be in yet another state. Namely, no corruption and no conflict detected and the party is able to open a ciphertext and compute f on clear. A corrupt party cannot be in this state since the honest parties would use consistent inputs and therefore the corrupt would not get access to conflicting encoded inputs that constitute the key of the ciphertexts. If any honest party is in this state, our protocol results in all parties outputting this output. In Round 3, this party can send the computed output along with the opening of the shares he recovered via the ciphertexts as 'proof' to convince the honest party of the validity of the output. The protocol Fair appears in Figure 3.3 and the schematic diagram is given in Section 3.7.1.1.

We now prove the correctness of Fair.

Protocol Fair

Inputs: Party P_i has x_i for $i \in [3]$.

Output: $y = f(x_1, x_2, x_3)$ or $\perp$ where the inputs and the function output belong to $\{0,1\}^\ell$.

Subprotocols: Fair_i for $i \in [3]$ (Figure 3.1), Cert_i for $i \in [3]$ (Figure 3.2), SKE $(\mathsf{Enc}, \mathsf{Dec})$ with 'special correctness' (Definition 2.10).

Round 1: For $i \in [3]$ and for distinct indices $j, k \in [3] \setminus \{i\}$

- Each P_i computes an encoding of length ℓs corresponding to its input x_i. For each bit b of x_i, the encoding $b_1, \ldots b_s$ is such that $b = \oplus_{\alpha=1}^{s} b_\alpha$. Reusing the notation, we refer to this encoding as P_i's input x_i and its length by ℓ.
- **Round 1** of Cert_i is run.
- **Round 1** of Fair_i are run with the following amendments: **(1)** The circuit in Fair_i is changed as follows: each input wire is replaced by a gate whose input consists of s new input wires and whose output is the exclusive-or of these wires. **(2)** P_j and P_k work with the permutation strings p_{jj} and p_{kk} respectively as x_{jk} and x_{kj}.
- P_i samples pp_i, generates $(\mathrm{c}_{ij}, \mathrm{o}_{ij}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, x_{ij})$, $(\mathrm{c}_{ik}, \mathrm{o}_{ik}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, x_{ik})$ and sends $\{\mathsf{pp}_i, \mathrm{c}_{ij}, \mathrm{c}_{ik}\}$ to P_j, P_k. Additionally, P_i sends o_{ij}, o_{ik} to P_j, P_k respectively.
- (Local Computation by P_i) P_i adds P_ℓ in $\mathcal{C}_i$ if $\mathsf{Open}(\mathrm{c}_{li}, \mathrm{o}_{li}) \neq x_{li}$. P_j adds P_k in $\mathcal{C}_j$ if: **(a)** p_{kk} not taken as x_{kj} or **(b)** the check in Fair_i or Cert_i fails. P_k adds P_j in $\mathcal{C}_k$ if: **(a)** p_{jj} not taken as x_{jk} or **(b)** the check in Fair_i or Cert_i fails.

Round 2: For $i \in [3]$ and for distinct indices $j, k \in [3] \setminus \{i\}$:

- If $P_i \notin \mathcal{C}_j$, P_j sends $(\mathsf{pp}_i, \mathrm{c}_{ij}, \mathrm{c}_{ik})$ to P_k. If $P_i \notin \mathcal{C}_k$, P_k sends $(\mathsf{pp}_i, \mathrm{c}_{ij}, \mathrm{c}_{ik})$ to P_j. They set $\mathsf{flag}_i = 1$ in case of mismatch or no communication.
- If $P_i \notin \mathcal{C}_j$, P_j participates in Cert_i as a garbler with input γ_j as $\{\mathcal{D}_i^j, \mathcal{D}_i^k, \mathcal{W}_k, \mathsf{pp}_i, \mathrm{c}_{ij}, \mathrm{c}_{ik}\}$ where $\mathcal{D}_i^j, \mathcal{D}_i^k$, $\mathcal{W}_k$ and $(\mathsf{pp}_i, \mathrm{c}_{ij}, \mathrm{c}_{ik})$ was received from P_i during Round 1 of $\mathsf{Fair}_j, \mathsf{Fair}_k$, Cert_k (assuming $k = (i+1) \mod 3$) and Fair respectively. Similar step is taken by P_k.
- If $\mathbf{cert}_i = (\gamma'_j, \gamma'_k)$, P_i sets $\mathcal{C}_i = P_l$ if $\gamma'_l \neq \{\mathcal{D}_i^j, \mathcal{D}_i^k, \mathcal{W}_k, \mathsf{pp}_i, \mathrm{c}_{ij}, \mathrm{c}_{ik}\}$ for $l \in \{j, k\}$.
- If $P_i \notin \mathcal{C}_j$, P_j participates in **Round 2** of Fair_i. When $P_k \notin \mathcal{C}_j$, P_j additionally sends the ciphertexts $\mathsf{ct}_{j\alpha}^{\beta}$ for $\beta \in \{0,1\}$ and $\alpha \in [\ell]$ created as follows. Let $\{\mathsf{X}^0_{l(\ell+\alpha)}, \mathsf{X}^1_{l(\ell+\alpha)}\}$, denote the encoding information of co-garbler P_k's input wire α corresponding to C_l $(l \in \{j, k\})$. Then

$\mathsf{ct}^{\beta}_{j\alpha} = \mathsf{Enc}_{\mathsf{sk}^{\beta}_{\alpha}}(\mathrm{o}_{jk}, \mathrm{o}_{kj})$ for $\mathsf{sk}^0_\alpha = \mathsf{X}^0_{j(\ell+\alpha)} \oplus \mathsf{X}^1_{k(\ell+\alpha)}$ and $\mathsf{sk}^1_\alpha = \mathsf{X}^1_{j(\ell+\alpha)} \oplus \mathsf{X}^0_{k(\ell+\alpha)}$. P_k takes similar steps.

– (Local Computation by P_i) Include P_l in $\mathcal{C}_i$ if $m_{ll} \neq x_{li}$ for $l \in \{j,k\}$. If $\mathcal{C}_i = \emptyset, \mathsf{flag}_j = 0, \mathsf{flag}_k = 0$, then use key $\mathsf{X}^{m^\alpha_{jk}}_{j(\ell+\alpha)} \oplus \mathsf{X}^{m^\alpha_{kk}}_{k(\ell+\alpha)}$ ($\alpha \in [\ell]$) to decrypt the ciphertexts $\mathsf{ct}^0_{j\alpha}$ or $\mathsf{ct}^1_{j\alpha}$ obtained from P_j. If the decryption succeeds, retrieve $\mathrm{o}_{kj}, \mathrm{o}_{jk}$. Execute $x_{kj} \leftarrow \mathsf{Open}(\mathrm{c}_{kj}, \mathrm{o}_{kj})$ and $x_{jk} \leftarrow \mathsf{Open}(\mathrm{c}_{jk}, \mathrm{o}_{jk})$. If the opening succeeds, then evaluate f on $(x_i, x_{ji} \oplus x_{jk}, x_{ki} \oplus x_{kj})$ to obtain y. Similarly, steps are taken with respect to P_j's input, using the key $\mathsf{X}^{m^\alpha_{jj}}_{j\alpha} \oplus \mathsf{X}^{m^\alpha_{kj}}_{k\alpha}$ to decrypt the ciphertexts $\mathsf{ct}^0_{k\alpha}$ or $\mathsf{ct}^1_{k\alpha}$ obtained from P_k.

A party P_i is said to be in $\mathtt{st}_\alpha$ for $\alpha \in [4]$ if the following conditions are satisfied. Let $(\mathsf{Y}_i, C^{\mathsf{dec}}_j, C^{\mathsf{dec}}_k)$, O^{dec}_j and O^{dec}_k denote the output of P_i in Fair_i, Fair_j and Fair_k, respectively. Let $\mathbf{cert}_i$, pad_j, and pad_k denotes the output of P_i in Cert_i, Cert_j and Cert_k respectively.

(i) $\mathtt{st}_1$(output is already computed): If y and proofs $(\mathrm{o}_{jk}, \mathrm{o}_{kj})$ are computed in Round 2.

(ii) $\mathtt{st}_2$ (no corruption and no conflict detected): If $((\mathcal{C}_i = \emptyset) \wedge (\mathsf{flag}_j = 0) \wedge (\mathsf{flag}_k = 0))$ (which implies $\mathsf{Y}_i \neq \perp$ and $\mathbf{cert}_i \neq \perp$)

(iii) $\mathtt{st}_3$ (corruption detected): If $(\mathcal{C}_i \neq \emptyset)$

(iv) $\mathtt{st}_4$ (conflict detected, but no corruption detected): If $(\mathsf{flag}_j = 1) \vee (\mathsf{flag}_k = 1)$

Round 3: Each P_i for $i \in [3]$ does the following based one of the four states that it belongs to.

– If in $\mathtt{st}_1$, then send y to P_j, P_k. Send o_{jk} to P_j and o_{kj} to P_k as proofs.

– If in $\mathtt{st}_2$, then send $(\mathsf{Y}_i, \mathbf{cert}_i, O^{\mathsf{dec}}_l)$ to P_l for $l \in \{j,k\}$.

– If in $\mathtt{st}_3$, then send O^{dec}_l to P_l for $l \in \{j,k\}$ only if $P_l \notin \mathcal{C}_i$.

– If in $\mathtt{st}_4$, then send $z_l = \mathsf{Enc}_{\mathsf{pad}_l}(O^{\mathsf{dec}}_l)$ to P_l only if $\mathsf{flag}_l = 1$. If $\mathsf{flag}_j = 1$ and $\mathbf{cert}_j$ received from P_j is same as pad_j, then set $\mathcal{C}_i = P_k$. Similar steps are taken to check and identify if P_j is corrupt. Update state from $\mathtt{st}_4$ to $\mathtt{st}_3$ if corrupt is identified.

– If in $\mathtt{st}_1$, then output y.

– If in $\{\mathtt{st}_2, \mathtt{st}_3, \mathtt{st}_4\}$ and if any other party is identified to be in $\mathtt{st}_1$, namely if y is received from P_j or P_k with o_{ki} or o_{ji} respectively such that $\mathsf{Open}(\mathsf{pp}_i, \mathrm{c}_{li}, \mathrm{o}_{li}) \neq \perp$ for $l \in \{j,k\}$, then output the received y.

– If in $\mathtt{st}_2$, then compute y as follows: Retrieve O^{dec}_i from either z_i (with $\mathbf{cert}_i$ as the key) received from P_j or from direct communication of P_j. If $\mathsf{d} \leftarrow \mathsf{eOpen}(\mathsf{epp}_k, C^{\mathsf{dec}}_k, O^{\mathsf{dec}}_i)$ is not $\perp$, then use

d to compute $y \leftarrow \mathsf{De}(\mathsf{Y}_i^k, \mathsf{d})$. Similar steps are executed with respect to P_k's communication if y is not computed yet.

– If in $\mathtt{st}_3$, then output $y \leftarrow \mathsf{De}(\mathsf{Y}_l^i, \mathsf{d})$ where Y_l is received from (honest) $P_l \notin \mathcal{C}_i$ and decoding information d is known as garbler during Fair_l. Otherwise output $y = \perp$.

– If in $\mathtt{st}_4$, output $y = \perp$.

Figure 3.3: A Three-Round Fair 3PC protocol

Lemma 3.13 *During* Fair, $P_j \notin \mathcal{C}_i$ *holds for honest* P_i, P_j.

Proof: An honest P_i will not include P_j in its corrupt set in the sub-protocols $\{\mathsf{Fair}_\alpha, \mathsf{Cert}_\alpha\}_{\alpha\in[3]}$ following Lemma 3.9, Lemma 3.11. Now we prove the statement individually investigating the three rounds of Fair.

In Round 1 of Fair, P_i includes P_j as corrupt only if (a) P_i, P_j are garblers and P_j sets $p_{jj} \neq x_{ji}$ or (b) P_j sends $\mathsf{pp}_j, \mathrm{c}_{ji}, \mathrm{o}_{ji}, x_{ji}$ to P_i such that $\mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{ji}, \mathrm{o}_{ji}) \neq x_{ji}$. None of them will be true for an honest P_j. In Round 2 of Fair, P_i includes P_j as corrupt only if (a) P_j is a garbler and P_i is an evaluator and $m_{jj} \neq x_{ji}$ or (b) P_i obtains $\mathbf{cert}_i = (\gamma'_j, \gamma'_k)$ and detects P_j's input γ'_j in Cert_i to be different from the information sent by him. The former will not be true for an honest P_j. The latter also cannot hold for honest P_j by correctness of the privacy-free garbling used. In the last round of Fair, P_i will identify P_j as corrupt, if it has $\mathsf{flag}_k = 1$ and yet receives $\mathbf{cert}_k$ which is same as pad_k from P_k. A corrupt P_k receives pad_k only by handing out correct and consistent common information to P_i and P_j until the end of Round 1. Namely, the following must be true for P_k to obtain pad_k (except for the case when it breaks the authenticity of the GC): (i) γ_i and γ_j for Cert_k must be same and (ii) P_k must not be in the corrupt set of any honest party at the end of Round 1. In this case, flag_k cannot be 1. □

Lemma 3.14 *No corrupt party can be in* $\mathtt{st}_1$ *by the end of Round 1, except with negligible probability.*

Proof: For a corrupt P_k, its honest garblers P_i and P_j creates the ciphertexts cts using keys with opposite meaning for their respective inputs from their garbled circuits. Since honest P_i and P_j use the same input for both the circuits, P_k will not have a key to open any of the ciphertexts. The openings $(\mathrm{o}_{ij}, \mathrm{o}_{ji})$ are therefore protected due to the security of the encryption scheme. Subsequently, P_k cannot compute y. □

Definition 3.1 *A party P_i is said to be 'committed' to a unique input x_i, if P_j holds $(\mathrm{c}_{ij}, \mathrm{c}_{ik}, \mathrm{o}_{ij}, x_{ij})$ and P_k holds $(\mathrm{c}_{ij}, \mathrm{c}_{ik}, \mathrm{o}_{ik}, x_{ik})$ such that:* **(a)** *$x_i = x_{ij} \oplus x_{ik}$ and* **(b)** *c_{ij} opens to x_{ij} via o_{ij} and likewise, c_{ik} opens to x_{ik} via o_{ik}.*

We next prove that a corrupt party must have committed its input if some honest party is in $\mathtt{st}_1$ or $\mathtt{st}_2$. To prove correctness, the next few lemmas then show that an honest party computes its output based on its own output or encoded output if it is in $\mathtt{st}_1$ or $\mathtt{st}_2$ or relies on the output or encoded output of the other *honest* party. In all cases, the output will correspond to the committed input of the corrupt party.

Lemma 3.15 *If an honest party is in $\{\mathtt{st}_1, \mathtt{st}_2\}$, then corrupt party must have committed a unique input.*

Proof: An honest P_i is in $\{\mathtt{st}_1, \mathtt{st}_2\}$ only when $\mathcal{C}_i = \emptyset$, $\mathsf{flag}_j = 0, \mathsf{flag}_k = 0$ hold at the end of Round 2. Assume P_k is corrupt. P_k has not committed to a unique x_k implies either it has distributed different copies of commitments $(\mathrm{c}_{ki}, \mathrm{c}_{kj})$ to the honest parties or distributed incorrect opening information to some honest party. In the former case, flag_k will be set by P_i. In the latter case, at least one honest party will identify P_k to be corrupt by the end of Round 1. If it is P_i, then $\mathcal{C}_i \neq \emptyset$. Otherwise, P_j populates its corrupt set with P_k, leading to P_i setting $\mathsf{flag}_k = 1$ in Round 2. □

Lemma 3.16 *If an honest party is in $\mathtt{st}_1$, then its output y corresponds to the unique input committed by the corrupt party.*

Proof: An honest P_i is in $\mathtt{st}_1$ only when $\mathcal{C}_i = \emptyset$, $\mathsf{flag}_j = 0, \mathsf{flag}_k = 0$ hold at the end of Round 2 and it computes y via decryption of the ciphertexts ct sent by either P_j or P_k. Assume P_k is corrupt. By Lemma 3.15, P_k has committed to its input. The condition $\mathsf{flag}_j = 0$ implies that P_k exchanges the commitments on the shares of P_j's input, namely $\{\mathrm{c}_{ji}, \mathrm{c}_{jk}\}$, honestly. Now if P_i opens honest P_j's ciphertext, then it unlocks the opening information for the missing shares, namely $(\mathrm{o}_{kj}, \mathrm{o}_{jk})$ corresponding to common and agreed commitments $(\mathrm{c}_{kj}, \mathrm{c}_{jk})$. Using these it opens the missing shares $x_{kj} \leftarrow \mathsf{Open}(\mathrm{c}_{kj}, \mathrm{o}_{kj})$ and $x_{jk} \leftarrow \mathsf{Open}(\mathrm{c}_{jk}, \mathrm{o}_{jk})$ and finally computes output on $(x_i, x_{ji} \oplus x_{jk}, x_{ki} \oplus x_{kj})$. Next, we consider the case when P_i computes y by decrypting a ct sent by corrupt P_k. In this case, no matter how the ciphertext is created, the binding property of NICOM implies that P_k will not be able to open $\mathrm{c}_{jk}, \mathrm{c}_{kj}$ to anything other than x_{jk}, x_{kj} except with negligible probability. Thus, the output computed is still as above and the claim holds. □

Lemma 3.17 *If an honest party is in* $\mathtt{st}_2$*, then its encoded output* Y *corresponds to the unique input committed by the corrupt party.*

Proof: An honest P_i is in $\mathtt{st}_2$ only when $\mathcal{C}_i = \emptyset$, $\mathsf{flag}_j = 0, \mathsf{flag}_k = 0$ hold at the end of Round 2. The conditions also imply that P_i has computed Y_i successfully (due to Lemma 3.10) and P_k has committed to its input (due to Lemma 3.15). Now we show that Y_i correspond to the unique input committed by the corrupt P_k. We first note that P_k must have used the same input for both the circuits C_j and C_k in Fair_i. Otherwise one of the ciphertexts prepared by honest P_j must have been opened and y would be computed, implying P_i belongs to $\mathtt{st}_1$ and not in $\mathtt{st}_2$ as assumed. We are now left to show that the input of P_k for its circuit C_k in Fair_i is the same as the one committed.

In Fair, honest P_j would use permutation string $p_{kk} = x_{kj}$ for permuting the commitments in $\mathcal{D}_k$ corresponding to x_k. Therefore, one can conclude that the commitments in $\mathcal{D}_k$ are constructed correctly and ordered as per x_{kj}. Now the only way P_k can decommit x'_k is by giving $m_{kk} = p_{kk} \oplus x'_k$. But in this case honest P_i would add P_k to $\mathcal{C}_i$ as the check $m_{kk} = x_{ki}$ would fail ($m_{kk} = p_{kk} \oplus x'_k \neq p_{kk} \oplus x_k$) and will be in $\mathtt{st}_3$ and not in $\mathtt{st}_2$ as assumed.

□

Lemma 3.18 *If an honest party is in* $\mathtt{st}_2$*, then its output* y *corresponds to the unique input committed by the corrupt party.*

Proof: Note that an honest party P_i in $\mathtt{st}_2$ either uses y of another party in $\mathtt{st}_1$ or computes output from its encoded output Y_i. The proof for the former case goes as follows. By Lemma 3.14, a corrupt P_k can never be in $\mathtt{st}_1$. The correctness of y computed by an honest P_j follows directly from Lemma 3.16. For the latter case, Lemma 3.17 implies that Y_i corresponds to the unique input committed by the corrupt party. All that needs to be ensured is that P_i gets the correct decoding information. The condition $\mathsf{flag}_j = \mathsf{flag}_k = 0$ implies that the commitment to the decoding information is computed and distributed correctly for both C_j and C_k. Now the binding property of eNICOM ensures that the decoding information received from either P_j (for C_k) or P_k (for C_j) must be correct implying correctness of y (by correctness of the garbling scheme). □

Lemma 3.19 *If an honest party is in* $\mathtt{st}_3$ *or* $\mathtt{st}_4$*, then its output* y *corresponds to the unique input committed by the corrupt party.*

Proof: An honest party P_i in $\mathtt{st}_3$ either uses y of another party in $\mathtt{st}_1$ or computes output from encoded output Y_j of P_j who it identifies as honest. For the latter case note that an honest

P_j will never be identified as corrupt by P_i, due to Lemma 3.13. The claim now follows from Lemma 3.14, Lemma 3.16 and the fact that corrupt P_k cannot forge the 'proof' o_{ij} (binding of NICOM) for the former case and from Lemma 3.17 and the fact that it possesses correct decoding information as a garbler for Y_j for the latter case. An honest party P_i in $\mathtt{st}_4$ only uses y of another party in $\mathtt{st}_1$. The lemma follows in this case via the same argument as before. □

Theorem 3.3 *Protocol* Fair *is correct.*

Proof: In order to prove the theorem, we show that if an honest party, say P_i outputs y that is not $\perp$, then it corresponds to x_1, x_2, x_3 where x_j is the input committed by P_j (Definition 3.1). We note that an honest P_i belong to one among $\{\mathtt{st}_1, \mathtt{st}_2, \mathtt{st}_3, \mathtt{st}_4\}$ at the time of output computation. The proof now follows from Lemmas 3.15,3.16,3.18,3.19. □

The property of fn implies: (a) if a corrupt party gets the output then so does the honest parties; (b) if an honest party gets the output then so does the other parties. We give the intuition for both below starting with (a). The formal proof appears in Section 3.7.1.2.

A corrupt P_k cannot be in $\mathtt{st}_1$ (due to Lemma 3.14). The only way it can retrieve the output is by having an honest party in $\mathtt{st}_1$ or $\mathtt{st}_2$. An honest party in $\mathtt{st}_3$ only releases the decoding information and it never release it to a corrupt party (Lemma 3.13 implies it identifies the honest party correctly). An honest party in $\mathtt{st}_4$ releases the encrypted decoding information z_k under key pad_k to P_k conditionally when $\mathsf{flag}_k = 1$. The condition $\mathsf{flag}_k = 1$ implies that P_k must have distributed the common information incorrectly and so γ_i and γ_j are not same. This further implies $\mathbf{cert}_k$ is not same as pad_k and so P_k does not have access to the key to open z_k and cannot recover the decoding information. So the corrupt P_k getting the output implies that at least one honest party is in $\{\mathtt{st}_1, \mathtt{st}_2\}$. Lemma 3.15 implies that in this case, P_k must have committed to a unique input. By Lemma 3.16 and Lemma 3.18, the y and encoded output Y computed by any honest party in $\mathtt{st}_1$ and in $\mathtt{st}_2$ respectively will correspond P_k's committed input. Further, if P_k computes encoded output Y_k, it also correspond to P_k's committed input. So no matter how the corrupt party compute the output, it will be with respect to unique (x_1, x_2, x_3). We need to show that both honest parties receive the same output. This easily follows when at least one honest party is in $\mathtt{st}_1$. We now prove the lemma based on the following cases. (a) Both P_i, P_j are in $\mathtt{st}_2$: They receive the decoding information from each other on the clear and use their respective computed encoded output to compute the output y. (b) P_i is in $\mathtt{st}_2$ and P_j in $\mathtt{st}_3$: P_i uses the decoding information sent exclusively to him by P_j and decode the output as in the previous case. P_j uses the encoded output of P_i, Y_i and its decoding information (held as a garbler) to compute the output. (c) P_i is in $\mathtt{st}_2$ and P_j in $\mathtt{st}_4$: P_j must be in $\mathtt{st}_4$ because of $\mathsf{flag}_i = 1$. If $\mathsf{flag}_k = 1$, P_i will have the same status for this flag and would

belong to $\mathtt{st}_4$. Now since $\mathsf{flag}_i = 1$, P_j sends encryption of the decoding information z_i to P_i who can use $\mathbf{cert}_i$ to decrypt z_i and compute the output as in the previous two cases. P_j, on noting that $\mathsf{flag}_i = 1$, yet P_i obtained $\mathbf{cert}_i = \mathsf{pad}_i$, will identify P_k to be corrupt, upgrade to $\mathtt{st}_3$ and compute the output as in the previous case.

Next, we argue for part (b). For an honest party to compute the output y, at least one honest party must be in $\{\mathtt{st}_1, \mathtt{st}_2\}$. If both belong to $\{\mathtt{st}_3, \mathtt{st}_4\}$, then neither P_k has committed any input (due to Lemma 3.15) nor anyone gets the output. The latter follows by the argument below. An honest party in $\mathtt{st}_3$ only outputs based on the encoded output of the other honest party. But since the other honest party is in $\{\mathtt{st}_3, \mathtt{st}_4\}$, it will output $\perp$. An honest party in $\mathtt{st}_4$ outputs $\perp$, except for the case it finds one in $\mathtt{st}_1$ which is not true for both P_j and P_k (Lemma 3.14). The corrupt P_k does not get the output too following the fact that it cannot be in $\mathtt{st}_1$ (Lemma 3.14) and it does not receive decoding information from an honest party. An honest party P_i in $\mathtt{st}_3$ sends the decoding information only to the *identified* honest party. An honest party P_i in $\mathtt{st}_4$ may send the encrypted decoding information z_k under key pad_k to P_k when $\mathsf{flag}_k = 1$. But the condition $\mathsf{flag}_k = 1$ implies that P_k must have distributed the common information incorrectly and so γ_i and γ_j are not same. This further implies $\mathbf{cert}_k$ is not same as pad_k and so P_k does not have access to the key to open z_k and cannot recover the opening information. Now we are left to show that when at least one honest party is in $\{\mathtt{st}_1, \mathtt{st}_2\}$, then everyone gets the output. This already follows from the argument given for the other direction.

3.4 2-round 3PC with Unanimous Abort

This section presents a tight upper bound for 3PC achieving ua in the setting with pair-wise private channels and a broadcast channel. The impossibility of one-round protocol in the same setting follows from "residual function" attack [112]. Our result from Section 3.2.2 rules out the possibility of achieving unanimous abort in the absence of a broadcast channel in two rounds. This protocol can be used to yield a round-optimal fair protocol with broadcast (lower bound in Section 3.2.1) by application of the transformation of [130] that compiles a protocol with ua to fn via evaluating the circuits that compute shares (using error-correcting secret sharing) of the function output using the protocol with ua and then uses an additional round for reconstruction of the output.

In an attempt to build a protocol with ua, we note that any protocol with ua must be robust to any potential misbehaviour launched via the private communication in the second round. Simply because, there is no way to report the abort to the other honest party who may have seen honest behaviour from the corrupt party all along and has got the output, leading to sa.

Our construction achieves unanimity by leveraging the availability of the broadcast channel to abort when a corrupt behaviour is identified either in the first round or in the broadcast communication in the second round, and behaving robustly otherwise. In summary, if the corrupt party does not strike in the first round and in the broadcast communication of the second round, then our construction achieves robustness.

Turning to the garbled circuit based constructions such as the two-round protocol of [129] achieving sa or the composition of three copies of the sub-protocol Fair_i of Fair, we note that the second round private communication that involves encoding information for inputs is crucial for computing the output and cannot transit via broadcast because of input privacy breach. A bit elaborately, the transfer of the encoding information for the inputs of the garblers can be completed in the first round itself and any inconsistency can be handled via unanimous abort in the second round. However, a similar treatment for the encoding information of the shares of the evaluator seems impossible as they are transferred to garblers only in the first round. We get past this seemingly impossible task via a clever 'two-part release mechanism' for the encoding information of the shares of the evaluator. Details follow.

Similar to protocol Fair, we build our protocol UAbort upon three parallel executions of a sub-protocol UAbort_i ($i \in [3]$), each comprising of two rounds and with each party P_i enacting the role of the evaluator once. With Fair_i as the starting point, each sub-protocol UAbort_i allows the parties to reach agreement on whether the run was successful and the evaluator got the output or not. A flag flag_i is used as an indicator. The protocol UAbort then decides on unanimous abort if at least one of the flags from the three executions UAbort_i for $i \in [3]$ is set to true. Otherwise, the parties must have got the output. Input consistency checks ensure that the outputs are identical. Intra-execution input consistency is taken care by cheat-recovery mechanism (similar and simplified version of what protocol Fair uses), while inter-execution input consistency is taken care by the same trick that we use in our fair protocol. Now looking inside UAbort_i, the challenge goes back to finding a mechanism for the honest evaluator to get the output when a corrupt party behaves honestly in the first round and in the broadcast communication of the second round. In other words, its private communication in the second round should not impact robustness. This is where the 'two-part release mechanism' for the encoding information of the shares of the evaluator kicks in. It is realized by tweaking the function to be evaluated as $f(x_j, x_k, (z_j \oplus r_j) \oplus (z_k \oplus r_k))$ in the instance UAbort_i where P_i enacts the role of the evaluator. Here r_j, r_k denote random pads chosen by the garblers P_j, P_k respectively in the first round. The encoding information for these are released to P_i *privately* in the first round itself. Any inconsistent behaviour in the first round is detected, the flag is set and the the protocol exits with $\perp$ unanimously. Next, z_j and z_k are the offsets of these

random pads with the actual shares of P_i's input and are available only at the end of first round. The encoding information for these offsets and these offsets themselves are transferred via broadcast in the second round for public verification. As long as the pads are privately communicated, the offsets do not affect privacy of the shares of P_i's input. Lastly, note that the encoding information for a garbler's input for its own generated circuit can be transferred in the first round itself. This ensures that a corrupt garbler misbehaves either in the first round or in the broadcast communication in the second round or lets the evaluator get the output via its own GC. We describe execution UAbort_i, assuming input consistency. Entwining the three executions, tackling the input consistency and the final presentation of protocol UAbort are done next. Lastly, we present the security proof.

3.4.1 Protocol UAbort_i

With the goal to achieve agreement among the honest parties regarding whether the evaluator got the output or not, UAbort_i starts with Fair_i and makes the following changes. First, the broadcast channel is used to reach agreement on the commitments to GCs and the encoding information. Second, a garbling scheme with soft decoding property is used to allow immediate output decoding. Third, a garbler opens its encoded input for its own GC in the first round itself. In addition, we implement the two-part release mechanism for P_i's shares where apart from the garblers, P_i too broadcasts the offsets in the second round. A flag flag_i is used to keep track if a complaint is raised for the first round communication by broadcast in the second round or the offsets broadcasted in parallel by both P_i and respective garblers do not match or the opening of the encoded input for the offsets fails. When flag_i remains to be false for the honest parties, an honest P_i must be able to evaluate and output from the GC prepared by the corrupt garbler. Because, the commitments to that GC and encoding information has been scrutinized by the honest co-garbler, the encoded input of the corrupt party has been verified by the evaluator, the release of the encoded inputs for the shares of the evaluator has been verified publicly and the offsets themselves matched. Lastly, since the flag when set to be true by any honest party in the end of first round can be propagated to all in the second round and is only set based on the broadcasts in the second round, all honest parties exit UAbort_i with an agreement on flag_i. We now present our protocol in Figure 3.4 assuming input consistency and prove its properties needed later.

Protocol UAbort_i

Inputs: Party P_α has x_α for $\alpha \in [3]$.

Common Inputs: The circuit $C((x_j, r_j, z_j), (x_k, r_k, z_k), \perp)$ that computes $f(x_j, x_k, (z_j \oplus r_j) \oplus (z_k \oplus$

$r_k)$) such that $z_j \oplus r_j = x_{ij}$, $z_k \oplus r_k = x_{ik}$ and $x_{ij} \oplus x_{ik} = x_i$ and where the inputs belong to $\{0,1\}^\ell$. For distinct $i, j, k \in [3]$, P_i acts as the evaluator and (P_j, P_k) as the garblers.

Output: All parties output boolean flag_i, initially set to 0. P_i outputs (y_j, y_k).

Primitives: A correct, private and authentic garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ with soft decoding, an eNICOM $(\mathsf{eGen}, \mathsf{eCom}, \mathsf{eOpen}, \mathsf{Equiv})$, a PRG G and a NICOM $(\mathsf{Com}, \mathsf{Open})$

Round 1:

– P_i randomly secret shares his input x_i as $x_i = x_{ij} \oplus x_{ik}$ and sends x_{ij} to P_j and x_{ik} to P_k.

– P_l for $l \in \{j, k\}$ samples $s_l \in_R \{0,1\}^\kappa$, epp_l and pp_l for G, eNICOM and NICOM resp. and:

  ◦ compute garbled circuit $(\mathsf{C}_l, \mathsf{e}_l, \mathsf{d}_l) \leftarrow \mathsf{Gb}(1^\kappa, C)$ using randomness from $\mathsf{G}(s_l)$. Assume $\{\mathsf{K}^0_{l\alpha}, \mathsf{K}^1_{l\alpha}\}_{\alpha\in[\ell]}$, $\{\mathsf{K}^0_{l(\ell+\alpha)}, \mathsf{K}^1_{l(\ell+\alpha)}\}_{\alpha\in[\ell]}$, $\{\mathsf{K}^0_{l(2\ell+\alpha)}, \mathsf{K}^1_{l(2\ell+\alpha)}\}_{\alpha\in[2\ell]}$, $\{\mathsf{K}^0_{l(4\ell+\alpha)}, \mathsf{K}^1_{l(4\ell+\alpha)}\}_{\alpha\in[2\ell]}$ correspond to the encoding information for the input of P_j, P_k (i.e. x_j, x_k), the random inputs chosen by P_j, P_k (i.e. r_j, r_k) and the offsets between the random pads and the secret shares of P_i (i.e. z_j, z_k) respectively.

  ◦ compute commitment for the GC as $(\mathrm{c}_l, \mathrm{o}_l) \leftarrow \mathsf{eCom}(\mathsf{epp}_l, \mathsf{C}_l)$

  ◦ sample permutation strings $p_{lj}, p_{lk} \in_R \{0,1\}^\ell$ for the inputs of P_j and P_k and compute commitments of encoding information as: For $b \in \{0,1\}$, $(\mathrm{c}^b_{l\alpha}, \mathrm{o}^b_{l\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}_l, \mathsf{e}^{p^\alpha_{lj}\oplus b}_{l\alpha})$, $(\mathrm{c}^b_{l(\ell+\alpha)}, \mathrm{o}^b_{l(\ell+\alpha)}) \leftarrow \mathsf{Com}(\mathsf{pp}_l, \mathsf{e}^{p^\alpha_{lk}\oplus b}_{l(\ell+\alpha)})$ when $\alpha \in [\ell]$, $(\mathrm{c}^b_{l(2\ell+\alpha)}, \mathrm{o}^b_{l(2\ell+\alpha)}) \leftarrow \mathsf{Com}(\mathsf{pp}_l, \mathsf{e}^b_{l(2\ell+\alpha)})$, $(\mathrm{c}^b_{l(4\ell+\alpha)}, \mathrm{o}^b_{l(4\ell+\alpha)}) \leftarrow \mathsf{Com}(\mathsf{pp}_l, \mathsf{e}^b_{l(4\ell+\alpha)})$ when $\alpha \in [2\ell]$.

  ◦ broadcast $\mathcal{D}_l = (\mathsf{epp}_l, \mathsf{pp}_l, \mathrm{c}_l, \{\mathrm{c}^b_{l\alpha}\}_{\alpha\in[6\ell],b\in\{0,1\}})$ and send $\{s_l, p_{lj}, p_{lk}, \mathrm{o}_l, \{\mathrm{o}^b_{l\alpha}\}_{\alpha\in[6\ell],b\in\{0,1\}}\}$ privately to the co-garbler $P_{\{j,k\}\setminus l}$.

– P_j computes indicator string $m_{jj} = p_{jj} \oplus x_j$, picks its share of pad $r_j \in_R \{0,1\}^\ell$ and sends $\left(\{\mathrm{o}^{m^\alpha_{jj}}_{j\alpha}, \mathrm{o}^{r^\alpha_j}_{j(2\ell+\alpha)}\}_{\alpha\in[\ell]}, m_{jj}, r_j\right)$ to P_i. Similarly, P_k computes m_{kk}, picks r_k and sends to P_i $\left(\{\mathrm{o}^{m^\alpha_{kk}}_{k(\ell+\alpha)}, \mathrm{o}^{r^\alpha_k}_{k(3\ell+\alpha)}\}_{\alpha\in[\ell]}, m_{kk}, r_k\right)$.

– (Local Computation by garblers) P_j sets $\mathsf{flag}_i = 1$ if $\mathcal{D}_k$ and $\{s_k, p_{kj}, p_{kk}, \mathrm{o}_k, \{\mathrm{o}^b_{k\alpha}\}_{\alpha\in[6\ell],b\in\{0,1\}}\}$ received from P_k are not consistent. P_k performs similar steps with respect to the values received from P_j.

– (Local Computation by evaluator) P_i sets $\mathsf{flag}_i = 1$ if (a) the openings of the input labels sent by P_j fail to open some commitment in $\mathcal{D}_j$ i.e $\mathsf{Open}(\mathsf{pp}_j, \mathrm{c}^{m^\alpha_{jj}}_{j\alpha}, \mathrm{o}^{m^\alpha_{jj}}_{j\alpha}) = \perp$ or $\mathsf{Open}(\mathsf{pp}_j, \mathrm{c}^{r^\alpha_j}_{j(2\ell+\alpha)}, \mathrm{o}^{r^\alpha_j}_{j(2\ell+\alpha)}) = \perp$ for some $\alpha \in [\ell]$ OR (b) the openings for the input labels sent by P_k fail to open some commitment in $\mathcal{D}_k$.

Round 2:

– P_j broadcasts `abort` if $\mathsf{flag}_i = 1$. Else, it computes its indicator string $m_{kj} = p_{kj} \oplus x_j$ for P_k's circuit and the offset $z_j = x_{ij} \oplus r_j$, sends $\left(\mathtt{OK}, \mathrm{o}_k, \{\mathrm{o}_{k\alpha}^{m_{kj}^\alpha}, \mathrm{o}_{k(2\ell+\alpha)}^{r_j^\alpha}, \mathrm{o}_{k(4\ell+\alpha)}^{z_j^\alpha}\}_{\alpha\in[\ell]}, m_{kj}\right)$ privately to P_i and broadcasts $\mathcal{W}_j = \left(z_j, \{\mathrm{o}_{j(4\ell+\alpha)}^{z_j^\alpha}\}_{\alpha\in[\ell]}\right)$. P_k performs similar steps.

– P_i broadcasts `abort` if $\mathsf{flag}_i = 1$. Else, it broadcasts $z_j = x_{ij} \oplus r_j$ and $z_k = x_{ik} \oplus r_k$

– Every party sets $\mathsf{flag}_i = 1$ if **(a)** `abort` was received or sent via broadcast in Round 2 OR **(b)** either z_j broadcast by (P_j, P_i) or z_k broadcast by (P_k, P_i) do not match OR **(c)** $\mathcal{D}_j, \mathcal{W}_j$ is not consistent i.e $\mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{j(4\ell+\alpha)}^{z_j^\alpha}, \mathrm{o}_{j(4\ell+\alpha)}^{z_j^\alpha}) = \perp$ or similarly $\mathcal{D}_k, \mathcal{W}_k$ is not consistent.

– (Local Computation by P_i) Output $y_j = y_k = \perp$ if $\mathsf{flag}_i = 1$. Else, with respect to C_j:

  ◦ open $\mathsf{C}_j \leftarrow \mathsf{eOpen}(\mathsf{epp}_j, \mathrm{c}_j, \mathrm{o}_j)$ where the opening is received from P_k.

  ◦ open $\mathsf{X}_j^\alpha = \mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{j\alpha}^{m_{jj}^\alpha}, \mathrm{o}_{j\alpha}^{m_{jj}^\alpha})$, $\mathsf{R}_j^\alpha = \mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{j(2\ell+\alpha)}^{r_j^\alpha}, \mathrm{o}_{j(2\ell+\alpha)}^{r_j^\alpha})$, and $\mathsf{Q}_j^\alpha = \mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{j(4\ell+\alpha)}^{z_j^\alpha}, \mathrm{o}_{j(4\ell+\alpha)}^{z_j^\alpha})$, for the openings received from P_j.

  ◦ open $\mathsf{X}_k^\alpha = \mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{j(\ell+\alpha)}^{m_{jk}^\alpha}, \mathrm{o}_{j(\ell+\alpha)}^{m_{jk}^\alpha})$, $\mathsf{R}_k^\alpha = \mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{j(3\ell+\alpha)}^{r_k^\alpha}, \mathrm{o}_{j(3\ell+\alpha)}^{r_k^\alpha})$ and $\mathsf{Q}_k^\alpha = \mathsf{Open}(\mathsf{pp}_j, \mathrm{c}_{j(5\ell+\alpha)}^{z_k^\alpha}, \mathrm{o}_{j(5\ell+\alpha)}^{z_k^\alpha})$ for $\alpha \in [\ell]$, for openings are received from P_k.

  ◦ If any of the above openings fail, set $y_j = \perp$. Else set $\mathsf{X} = \mathsf{X}_j|\mathsf{X}_k|\mathsf{R}_j|\mathsf{R}_k|\mathsf{Q}_j|\mathsf{Q}_k$, run $\mathsf{Y}_j \leftarrow \mathsf{Ev}(\mathsf{C}_j, \mathsf{X})$ and $y_j \leftarrow \mathsf{sDe}(\mathsf{Y}_j)$.

  Similar steps as above with respect to C_k is executed to compute Y_k and y_k.

Figure 3.4: Protocol UAbort_i

Lemma 3.20 *At the end of protocol* UAbort_i*, all honest parties output the same* flag_i*.*

Proof: We have two cases based on whether atleast one honest party set $\mathsf{flag}_i = 1$ at the end of Round 1. If this is true, then the honest party would broadcast `abort` in Round 2 and all honest parties would output $\mathsf{flag}_i = 1$. Otherwise, an honest party sets flag_i based on the following conditions (a) `abort` was broadcast in Round 2 or (b) either z_j broadcast by (P_j, P_i) or z_k broadcast by (P_k, P_i) do not match or (c) $(\mathcal{D}_j, \mathcal{W}_j)$ or $(\mathcal{D}_k, \mathcal{W}_k)$ is inconsistent. All these checks are with respect to broadcast messages. Therefore, we can conclude that every honest party will output identical flag_i. □

Lemma 3.21 *Assuming input consistency, if* $\mathsf{flag}_i = 0$*, then* $y_k \neq \perp$ *where* P_k *is corrupt.*

Proof: First, Lemma 3.20 implies that both P_i, P_j output identical $\mathsf{flag}_i = 0$. Now $\mathsf{flag}_i = 0$ implies that: **(a)** C_k and the commitments to the encoding information are computed correctly;

(b) the opening of encoding information $\mathsf{X}_k, \mathsf{R}_k$ for C_k is correct in Round 1 with high probability due to binding property of eNICOM and NICOM; **(c)** the opening of the remaining encoding information Q_k is correct with high probability due to binding property of NICOM. P_j being honest would open the encoding relevant to his input for C_k, namely, $\mathsf{X}_j, \mathsf{R}_j, \mathsf{Q}_j$. So P_i has got complete encoded input X for C_k and will evaluate C_k to obtain y_k. Thus, if $\mathsf{flag}_i = 0$, then y_k will not be $\perp$. □

3.4.2 Protocol UAbort

Our two-round 3PC protocol UAbort achieving ua composes UAbort_i for $i \in [3]$ in parallel. Assuming input consistency, entwining the three executions requires tapping all the flags returned by the three executions and outputting the result computed as an evaluator when none of them are set to true and $\perp$, otherwise. This works since when a flag for an execution UAbort_i is false, then the evaluator P_i is guaranteed to get the output. The challenge that remains to handle is input consistency within and across executions which ensures the outputs computed are the same irrespective of the execution and GC. The inter-execution input consistency, i.e the consistency of the input committed by P_i in UAbort_i and the inputs given to the GCs constructed by P_i as garbler in the remaining two executions are enforced using the same trick that we use in Fair via setting the permutation strings as the shares of the parties' input.

Dealing with the input consistency within an execution UAbort_i to make sure the garblers provide the same input for both the GCs without inflating the round complexity constitutes yet another challenge. Noting that this misbehaviour has no way to show up in the common flag as this is targeted via the private communication in the second round, the evaluator must find a way to robustly compute the output when conflicted outputs are computed from the two garbled circuits. This output must be based on the input of the corrupt garbler that it has committed as an evaluator and received output based on. We use the trick of "proof-of-cheating" mechanism [145] to enable an (honest) evaluator with conflicting outputs to retrieve the inputs committed by both garblers in their respective instances. To be specific, the output keys corresponding to the mismatched output bit in the two garbled circuits, say C_1 and C_3 in UAbort_2, enables the evaluator P_2 to unlock the missing shares, namely, x_{31} and x_{13} of the two garblers from UAbort_3 and UAbort_1 respectively. To ensure that the recovered missing shares are as distributed in UAbort_1 and UAbort_3, the shares are committed via NICOM by the input owners and the openings are encrypted by the holders (as in Fair). The binding of NICOM, prevents a corrupt P_1 to lie on (x_{13}, x_{31}). This allows the honest party to compute the same output that P_1 gets from UAbort_1. Lastly, the flag in execution UAbort_i also takes into account consistent dealing of the commitments by its evaluator P_i. Our protocol appears in Figure

Figure 3.5, the proof of correctness and the proof of security below. We use Definition 3.1 for input commitment.

Protocol $\mathsf{UAbort}()$

Inputs: Party P_i has x_i for $i \in [3]$.

Output: $y = f(x_1, x_2, x_3)$ or $\perp$.

Sub-protocols: UAbort_i for $i \in [3]$ (Figure 3.4), a NICOM $(\mathsf{Com}, \mathsf{Open})$, CPA-secure SKE Enc.

Round 1: For $i \in [3]$ and for distinct indices $j, k \in [3] \setminus \{i\}$

– **Round 1** of UAbort_i are run parallel. In UAbort_i, P_j and P_k work with the permutation strings p_{jj} and p_{kk} respectively as x_{jk} and x_{kj}.

– P_i samples pp_i, generates $(\mathrm{c}_{ij}, \mathrm{o}_{ij}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, x_{ij})$, $(\mathrm{c}_{ik}, \mathrm{o}_{ik}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, x_{ik})$, broadcasts $\{\mathsf{pp}_i, \mathrm{c}_{ij}, \mathrm{c}_{ik}\}$ and sends o_{ij}, o_{ik} to P_j, P_k respectively.

– (Local Computation) P_i sets $\mathsf{flag}_i = 1$ if $\mathsf{Open}(\mathrm{c}_{li}, \mathrm{o}_{li}) \neq x_{li}$ or $m_{ll} \neq x_{li}$ for $l \in \{j, k\}$. P_j sets $\mathsf{flag}_i = 1$ if: **(a)** p_{kk} not taken as x_{kj} or **(b)** the check in UAbort_i fails. **(c)** $\mathsf{Open}(\mathrm{c}_{ij}, \mathrm{o}_{ij}) \neq x_{ij}$. P_k sets $\mathsf{flag}_i = 1$ if: **(a)** p_{jj} not taken as x_{jk} or **(b)** the check in UAbort_i fails. **(c)** $\mathsf{Open}(\mathrm{c}_{ik}, \mathrm{o}_{ik}) \neq x_{ik}$.

Round 2:

– **Round 2** of UAbort_i for $i \in [3]$ are run parallel. In UAbort_i, the garbler P_j (similar steps will be taken by P_k) does the following additionally if $\mathsf{flag}_i \neq 1$. Let $\{\mathsf{Y}_l^0, \mathsf{Y}_l^1\}$, denote the encoding information for output wire corresponding to C_l ($l \in \{j, k\}$). It sends two ciphertexts $(\mathsf{ct}_j^0, \mathsf{ct}_j^1)$ where $\mathsf{ct}_j^0 = \mathsf{Enc}_{\mathsf{Y}_j^0 \oplus \mathsf{Y}_k^1}(\mathrm{o}_{jk}, \mathrm{o}_{kj})$ and $\mathsf{ct}_j^1 = \mathsf{Enc}_{\mathsf{Y}_j^1 \oplus \mathsf{Y}_k^0}(\mathrm{o}_{jk}, \mathrm{o}_{kj})$.

– For $i \in [3]$, party P_i computes output as follows:

  ◦ If $\mathsf{flag}_\alpha = 1$ for some $\alpha \in [3]$, then output $y = \perp$.

  ◦ Otherwise, output y as y_j when $y_j = y_k$ or $y_k = \perp$, as y_k when $y_j = \perp$ where (y_j, y_k) are output from UAbort_i.

  ◦ Otherwise, let the encoded outputs corresponding to $\mathsf{C}_j, \mathsf{C}_k$ in UAbort_i are $\mathsf{Y}_j, \mathsf{Y}_k$. It uses key $\mathsf{Y}_j^{y_j} \oplus \mathsf{Y}_k^{y_k}$ to decrypt the ciphertext $\mathsf{ct}_j^{y_j}$ obtained from P_j to retrieve $(\mathrm{o}_{jk}, \mathrm{o}_{kj})$. It executes $x_{kj} \leftarrow \mathsf{Open}(\mathrm{c}_{kj}, \mathrm{o}_{kj})$ and $x_{jk} \leftarrow \mathsf{Open}(\mathrm{c}_{jk}, \mathrm{o}_{jk})$. If x_{kj} or $x_{jk} = \perp$, then they are recomputed as above using $\mathsf{ct}_k^{y_j}$ obtained from P_k. Then P_i evaluates f on inputs $(x_i, x_{ji} \oplus x_{jk}, x_{ki} \oplus x_{kj})$ to obtain y.

Figure 3.5: A Two-Round 3PC protocol achieving unanimous abort

Lemma 3.22 *If a corrupt party P_k has not committed its input or does not use the committed input in its GCs in $\{\mathsf{UAbort}_i, \mathsf{UAbort}_j\}$, then each honest party outputs $y = \perp$.*

Proof: P_k has not committed to a unique input implies it has not dealt correct opening to one or both the honest parties. In either case, `abort` is raised in the second round, leading to an output that is $\perp$. Now assume P_k uses input $x'_k \neq x_k$ during UAbort_i for its own GC. P_k should use x_{kj} as the permutation string p_{kk} in execution UAbort_i for permuting the commitments corresponding to x_k. If it does not, then honest P_j sets $\mathsf{flag}_i = 1$ in Round 1 and broadcasts `abort` in Round 2. Otherwise, the commitments are constructed correctly and ordered as per x_{kj}. Now the only way P_k can decommit x'_k is by giving $m_{kk} = p_{kk} \oplus x'_k$. But in this case honest P_i would set $\mathsf{flag}_i = 1$ in Round 1 and broadcast `abort` in Round 2 as the check $m_{kk} = x_{ki}$ would fail ($m_{kk} = p_{kk} \oplus x'_k \neq p_{kk} \oplus x_k$). Thus, every honest party outputs $y = \perp$. □

Theorem 3.4 *Protocol* UAbort *is correct.*

Proof: In order to prove the theorem, we show that if an honest party, say P_i outputs y that is not $\perp$, then it corresponds to (x_1, x_2, x_3) where x_j is the input committed by P_j. Assume that P_k is corrupt. Recall that P_i outputs y_j and y_k in UAbort_i on evaluating the GCs of the garblers P_j and P_k respectively. We have the following cases.

- $y = y_k$. Follows from Lemma 3.21, 3.22.
- $y \neq y_k$. In this case, $y \neq y_j$ either as y is set to y_j when $y_j = y_k$ or $y_k = \perp$. Following Lemma 3.21, y_k cannot be $\perp$. So it must be that P_i retrieves the output via opening the ciphertexts. If the output is computed just from the ciphertext of honest P_j, then y is computed as $f(x_i, x_{ji} \oplus x_{jk}, x_{ki} \oplus x_{kj})$ using openings o_{kj}, o_{jk} given by P_j. Since an honest P_j correctly reveals the opening o_{kj} of the share of P_k's input given to P_j and o_{jk} corresponding to his input share, $f(x_i, x_{ji} \oplus x_{jk}, x_{ki} \oplus x_{kj})$ corresponds to the correct value. If the output is computed from the ciphertext of corrupt P_k, then y computed must be still as above as a corrupt P_k cannot open the shares x_{jk}, x_{kj} in an incorrect way (following binding property of NICOM).

□

The intuition for achieving ua follows from the correctness and Lemma 3.20 that implies the honest parties will be on the same page for all flags. The formal proof appears in Section 3.7.2.

3.5 3-round 3PC with Guaranteed Output Delivery

In this section, we present a three-round 3PC protocol achieving god, given access to pairwise-private channels and a broadcast channel. The protocol is round-optimal following 3-round lower bound for fair 3PC proven in Section 3.2.1. The necessity of the broadcast channel for achieving god with strict honest majority follows from [67].

Our tryst starts with the known generic transformations that are relevant such as the transformations from the ua to idfair (identifiable fairness) protocol [130] or idfair to god [66]. However, these transformations being non-round-preserving do not turn out to be useful. Turning a 2-round protocol offering ua (or even sa) with identifiability (when the honest parties learn about the identity of the corrupt when deprived of the output) to a 3-round protocol with god in a black-box way show some promise. The third round can be leveraged by the honest parties to exchange their inputs and compute output on the clear. We face two obstacles with this approach. First, there is neither any known 2-round construction for sa / ua with identifiability nor do we see how to transform our ua abort protocol to one with identifiability in two rounds. Second, when none of the parties (including the corrupt) receive output from the sa/ua protocol and the honest parties compute it on the clear in the third round by exchanging their inputs and taking a default value for the input of the corrupt party, it is not clear how the corrupt party can obtain the same output (note that the ideal functionality demands delivering the output to the adversary).

We get around the above issues by taking a non-blackbox approach and tweaking UAbort_i and Fair_i to get yet another sub-protocol GOD_i that achieves a form of local identifiability. Namely, the evaluator P_i in GOD_i either successfully computes the output or identifies the corrupt party. As usual, our final protocol GOD is built upon three parallel executions of GOD_i ($i \in [3]$), each comprising of two rounds and with each party P_i enacting the role of the evaluator once. Looking ahead, the local identifiability helps in achieving god as follows. In a case when both honest parties identify the corrupt party and the corrupt party received the output by the end of Round 2, the honest parties can exchange their inputs and reconstruct the corrupt party's input using the shares received during one of the executions of GOD_i and compute the function on clear inputs in the third round. Otherwise, the honest party who identifies the corrupt can simply accept the output computed and forwarded by the other honest party. The issue of the corrupt party getting the same output as that of the honest parties when it fails to obtain any in its instance of GOD_i is taken care as follows. First, the only reason a corrupt party in our protocol does not receive its output in its instance of GOD_i is due to denial of committing its input. In this case it is detected early and the honest parties exchange inputs in

the second round itself so that at least one honest party computes the output using a default input of the corrupt party by the end of Round 2 and hands it over to others in Round 3.

In the following, we describe one execution GOD_i. Entwining the three executions, tackling the input consistency and the final presentation of protocol GOD are done next. The security proof appears in Section 3.7.3.

3.5.1 Protocol GOD_i

Recall that the goal of GOD_i for $i \in [3]$ comprising of two rounds, is either successful computation of output or successful identification of the corrupt party by the evaluator P_i. Starting with the ideas of UAbort_i, we note that UAbort_i only ensures detection of the corrupt party by *s*ome honest party that is not necessarily the evaluator in case of a failed output computation. Specifically, a garbler would identify his co-garbler to be corrupt when the broadcast communication of co-garbler is not consistent with the privately shared randomness. In such a case, the evaluator neither gets the output nor has any clue on the identity of the corrupt, which is not in accordance with the goal of GOD_i. In the absence of broadcast, Fair_i gives even weaker guarantee where the best any party gets to know is a conflict. The above is handled by having the garblers send their inputs on clear to the evaluator on finding inconsistent behaviour of the fellow garbler in the first round. If both the garblers are in conflict with each other, the evaluator gets their inputs and computes the function on clear. Otherwise, the evaluator can either evaluate at least one of the GCs or identify the corrupt. Lastly, as we do not require unanimity of any form at the end of two rounds, we simplify GOD_i by removing the two-part release mechanism and the flag altogether. Like UAbort_i, we do not take care of the possibility of a corrupt garbler handing out inconsistent input for the two GCs in GOD_i. This is taken care in the main protocol GOD via the input consistency. P_i outputs $(y = (y_j, y_k), \mathsf{Y}_i = (\mathsf{Y}_i^j, \mathsf{Y}_i^k), \mathcal{C}_i)$, the outputs computed from two GCs, the encoded outputs and its corrupt set, all initially set to $\perp$ and to be used in the main construction. If both (y_j, y_k) are $\perp$, then the corrupt set will be non-empty. The garblers output their corrupt set. We now prove a few lemmas. The protocol GOD_i appears in Figure 3.6.

Protocol GOD_i

Inputs: Party P_α has x_α for $\alpha \in [3]$.

Common Inputs: Same as Fair_i (Figure 3.1).

Output: A garbler P_l for $l \in \{j, k\}$ outputs $\mathcal{C}_l$. P_i outputs $(y = (y_j, y_k), \mathsf{Y}_i = (\mathsf{Y}_i^j, \mathsf{Y}_i^k), \mathcal{C}_i)$ where y is the output (initially set to $\perp$) and $\mathcal{C}_i$ denotes the corrupt set maintained by P_i.

Primitives: A garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ that is correct, private and authentic, an eNICOM $(\mathsf{eGen}, \mathsf{eCom}, \mathsf{eOpen}, \mathsf{Equiv})$ and a PRG G.

Round 1: Same as Round 1 of Fair_i (Figure 3.1) except that the garblers do not commit to the decoding information and $\mathcal{D}_l$ computed by garbler P_l ($l \in \{j,k\}$) is communicated via broadcast.

Round 2:

– P_j computes indicator string $m_{jj} = p_{jj} \oplus x_j, m_{kj} = p_{kj} \oplus x_j$. If $P_k \notin \mathcal{C}_j$, then send to P_i $\left(\mathtt{OK}, (\mathrm{o}_k, \{\mathrm{o}_{k\alpha}^{m_{kj}^\alpha}, \mathrm{o}_{k(2\ell+\alpha)}^{x_{ij}^\alpha}\}_{\alpha\in[\ell]}, m_{kj}), (\{\mathrm{o}_{j\alpha}^{m_{jj}^\alpha}, \mathrm{o}_{j(2\ell+\alpha)}^{x_{ij}^\alpha}\}_{\alpha\in[\ell]}, m_{jj})\right)$. Otherwise, it sends to P_i $\left(\mathtt{nOK}, x_j, (\{\mathrm{o}_{j\alpha}^{m_{jj}^\alpha}, \mathrm{o}_{j(2\ell+\alpha)}^{x_{ij}^\alpha}\}_{\alpha\in[\ell]}, m_{jj})\right)$. P_k performs similar steps.

– (Local Computation) If $\mathtt{nOK}$ is received from both P_j, P_k, then compute $y_j = y_k = f(x_1, x_2, x_3)$ using x_j, x_k. Otherwise, one of the parties has sent $\mathtt{OK}$. Assume, for simplicity that P_k has sent $\mathtt{OK}$. Then, compute Y_i^j as in Fair_i. If $\mathsf{Y}_i^j \neq \perp$, then $y_j \leftarrow \mathsf{sDe}(\mathsf{Y}_i^j)$. If P_j sent $\mathtt{OK}$, then similar steps as above for C_k will be executed and y_k will be set.

Figure 3.6: Protocol GOD_i

Lemma 3.23 *$P_\beta \notin \mathcal{C}_\alpha$ holds for honest P_α, P_β.*

Proof: An honest P_α would include P_β in $\mathcal{C}_\alpha$ only if one of the following holds: (a) Both P_α, P_β are garblers and P_β broadcasts $\mathcal{D}_\beta$ inconsistent with values privately shared with P_α (b) P_α is an evaluator and P_β is a garbler and P_β's opening of a committed encoded input or garbled circuit approved by him fails. It is easy to verify that the cases will never occur for honest (P_α, P_β). □

Lemma 3.24 *Assuming input consistency, at the end of protocol GOD_i, an honest evaluator P_i either computes the output or identifies the corrupt party.*

Proof: Assume that P_k is the corrupt garbler. We have two cases.

– **P_k sends $\mathtt{nOK}$:** If P_j sends $\mathtt{nOK}$ too, P_i receives x_l from P_l for $l \in \{j,k\}$ (else P_k is identified to be corrupt) and computes f on inputs x_i, x_j, x_k. If P_j sends $\mathtt{OK}$, then the garbled circuit C_k is correctly constructed and the corresponding encoding information is correctly committed. The only way a corrupt garbler P_k can stop P_i from evaluating C_k (and avoid being caught by P_i) is by sending encoded inputs corresponding to (x_k, x_{ik}) that are inconsistent with C_k via breaching the binding property of NICOM which happens only with negligible probability.

– P_k **sends** OK**:** In this case, the binding property of eNICOM ensures that with high probability the correct C_j is opened (otherwise P_k is caught). The arguments now follow as the previous case where the probability that P_i does not get the output and does not detect P_k reduces to the probability of breaching binding of NICOM.

□

3.5.2 Protocol GOD

Our three-round 3PC protocol achieving god composes GOD_i for $i \in [3]$, with each party acting as the evaluator in parallel. At a high level, the protocol assures that every party either outputs y that is not $\perp$ or identifies the corrupt by end of second round. In the third round, a party simply sends his output if it is non-$\perp$, else it sends its input and share of the corrupt party's input to the honest party alone. A party outputs its own output computed in second round if it is not $\perp$. Otherwise, it outputs the non-$\perp$ output received from the non-faulty party or computes the output using the input and share sent by the non-faulty party. The input consistency is handled exactly as in UAbort. Additionally every party maintains a corrupt set and populates it when it identifies the corrupt. The overall composition maintains god as below based on when a corrupt party chooses to expose itself.

The cases when a corrupt P_i is detected by the end of first round itself, the honest party who makes the identification, halts the execution where it plays the evaluator with the corrupt set as the output and also halts GOD_i to stop letting P_i get output in GOD_i. Since the detection may be owing to non-commitment of any input by P_i in GOD_i, the unique input of P_i has to be set to the one that it commits in the running execution or as a default value when either there is no running execution or P_i does not commit to anything in the running execution. Specifically, if both the honest parties identify P_i to be corrupt by the end of first round, both would have exchanged their input as per the code of GOD protocols and a default common value is taken as the input of P_i to compute the function output by the end of second round itself and the output is handed over to P_i in third round. Handing the output to corrupt P_i is necessary to technically realise the functionality correctly where the corrupt party also gets the output. If just one of the parties detects the corrupt party P_i, say P_j, it stops its execution as the evaluator in GOD_j and as garbler in GOD_i to prevent P_i getting any output in GOD_i. Now P_i has two options: either it passes on its input on clear to P_k or it lets P_k to evaluate the garbled circuit of P_j by giving its encoded input. In either case, this input of P_i is taken as his committed input and the output computed by P_k is the one to be outputted by all. (Note that P_i's own GC will not be approved by its co-garbler who has identified it as corrupt by the end of first round.) P_k can simply pass on the output to P_i and P_j in the third round and P_j

simply takes the output of P_k who it knows to be honest. Our protocol appears in Figure 3.7. The proof of correctness appear below and the full proof in Section 3.7.3.

Protocol GOD

Inputs: Party P_i has x_i for $i \in [3]$.

Output: $y = f(x_1, x_2, x_3)$.

Sub-protocols Used: GOD_i, $i \in [3]$ (Figure 3.6), a NICOM $(\mathsf{Com}, \mathsf{Open})$, CPA-secure SKE Enc.

Round 1: For $i \in [3]$ and for distinct indices $j, k \in [3] \setminus \{i\}$

- **Round 1** of GOD_i are run parallel. In GOD_i, P_j and P_k work with the permutation strings p_{jj} and p_{kk} respectively as x_{jk} and x_{kj}.
- P_i samples pp_i, generates $(\mathrm{c}_{ij}, \mathrm{o}_{ij}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, x_{ij})$, $(\mathrm{c}_{ik}, \mathrm{o}_{ik}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, x_{ik})$, broadcasts $\{\mathsf{pp}_i, \mathrm{c}_{ij}, \mathrm{c}_{ik}\}$ and sends o_{ij}, o_{ik} to P_j, P_k respectively.
- (Local Computation) P_i adds P_ℓ in $\mathcal{C}_i$ if $\mathsf{Open}(\mathrm{c}_{li}, \mathrm{o}_{li}) \neq x_{li}$. P_j adds P_k in $\mathcal{C}_j$ if: **(a)** p_{kk} not taken as x_{kj} or **(b)** the check in GOD_i fails. P_k adds P_j in $\mathcal{C}_k$ if: **(a)** p_{jj} not taken as x_{jk} or **(b)** the check in GOD_i fails.

Round 2: For $i \in [3]$ and for distinct indices $j, k \in [3] \setminus \{i\}$

- If $P_i \notin \mathcal{C}_j$, then P_j participates in GOD_i. If $P_k \notin \mathcal{C}_j$, it additionally sends the following ciphertexts $\{\mathsf{ct}_j^0, \mathsf{ct}_j^1\}$ created as below. Let $\{\mathsf{Y}_l^0, \mathsf{Y}_l^1\}$, denote the encoding information for output wire corresponding to C_l ($l \in \{j, k\}$). Then $\mathsf{ct}_j^0 = \mathsf{Enc}_{\mathsf{Y}_j^0 \oplus \mathsf{Y}_k^1}(\mathrm{o}_{jk}, \mathrm{o}_{kj})$ and $\mathsf{ct}_j^1 = \mathsf{Enc}_{\mathsf{Y}_j^1 \oplus \mathsf{Y}_k^0}(\mathrm{o}_{jk}, \mathrm{o}_{kj})$.
- P_i includes P_l in $\mathcal{C}_i$ if $m_{ll} \neq x_{li}$ for $l \in \{j, k\}$.
- (Local Computation by P_i) If $\mathcal{C}_i = \emptyset$, then compute $y = (y_j, y_k)$ as in GOD_i. If $y_j = y_k$ ($\neq \perp$) or one of them is non-$\perp$, set y to one of them in the former and to the not-$\perp$ in the latter. If $y_j \neq y_k$, use key $\mathsf{Y}_j^{y_j} \oplus \mathsf{Y}_k^{y_k}$ to decrypt the ciphertexts $\mathsf{ct}_j^{y_j}$ obtained from P_j to retrieve $(\mathrm{o}_{jk}, \mathrm{o}_{kj})$. Execute $x_{kj} \leftarrow \mathsf{Open}(\mathrm{c}_{kj}, \mathrm{o}_{kj})$ and $x_{jk} \leftarrow \mathsf{Open}(\mathrm{c}_{jk}, \mathrm{o}_{jk})$. If x_{kj} or $x_{jk} = \perp$, then they are recomputed as above using $\mathsf{ct}_k^{y_j}$ obtained from P_k. Then evaluate f on inputs $(x_i, x_{ji} \oplus x_{jk}, x_{ki} \oplus x_{kj})$ to obtain y. If $\mathcal{C}_i \neq \emptyset$, $y = \perp$ and P_i receives x_l from $P_l \notin \mathcal{C}_i$, compute y as the value of f on x_i, x_l and a default value for the remaining party's input.

Round 3: Each P_i for $i \in [3]$ either has $y \neq \perp$ or $\mathcal{C}_i \neq \emptyset$. It does the following

- If $y \neq \perp$, send y to P_j, P_k. Send (x_i, x_{ji}) to P_k when $P_j \in \mathcal{C}_i$ or (x_i, x_{ki}) to P_j when $P_k \in \mathcal{C}_i$.

– If $y \neq \perp$, output y. Else if $P_l \notin \mathcal{C}_i$ sends y, output y. Else if $P_j \notin \mathcal{C}_i$ sends (x_j, x_{kj}), then compute y as the output of f on $(x_i, x_j, x_{kj} \oplus x_{ki})$. Similar steps are executed when $P_k \notin \mathcal{C}_i$ and it sends (x_k, x_{jk}) i.e y is derived from $(x_i, x_k, x_{jk} \oplus x_{ji})$.

Figure 3.7: A Three-Round 3PC protocol achieving god

Lemma 3.25 *$P_\beta \notin \mathcal{C}_\alpha$ holds for honest P_α, P_β in protocol GOD_i, where $i \in [3]$.*

Proof: This lemma follows from Lemma 3.23 and the fact that the following will not be true for honest (P_α, P_β): (a) P_β sends $\mathrm{o}_{\beta\alpha}, x_{\beta\alpha}$ to P_α such that $\mathsf{Open}(c_{\beta\alpha}, \mathrm{o}_{\beta\alpha}) \neq x_{\beta\alpha}$ (b) Both P_α, P_β are garblers and $p_{\beta\beta} \neq x_{\beta\alpha}$. (c) P_β is the garbler, P_α is an evaluator and $m_{\beta\beta} \neq x_{\beta\alpha}$ □

Lemma 3.26 *Every party P_i uses its 'committed' input x_i (Definition 3.1) in its GCs in $\{\mathsf{GOD}_j, \mathsf{GOD}_k\}$. Otherwise, it is identified by at least one of the honest parties.*

Proof: P_i has not committed to its input implies it has not dealt correct opening to one or both the honest parties. In either case, at least one of the honest parties identify him. Now assume P_i has committed to input x_i but uses input $x_i' \neq x_i$ during GOD_j for the garbled circuit constructed by P_i. P_i should use x_{ik} as the permutation string p_{ii} in execution GOD_j for permuting the commitments corresponding to x_i. If P_i does otherwise, then it is identified by honest P_k. Otherwise, the commitments are constructed correctly and ordered as per x_{ik}. Now the only way P_i can decommit x_i' is by giving $m_{ii} = p_{ii} \oplus x_i'$. But P_j identifies P_i as corrupt as $m_{ii} = p_{ii} \oplus x_i' \neq p_{ii} \oplus x_i$. □

We now prove correctness of the protocol accounting exhaustively all the scenarios: the corrupt party

– belongs to the corrupt set of both the honest parties,

– belongs to the corrupt set of exactly one of the honest parties and

– does not belong to the corrupt set of the honest parties

by the end of the first round. For simplicity, we assume that P_k is the corrupt party and P_i, P_j are the honest parties.

Lemma 3.27 *Assuming that the corrupt party belongs to the corrupt set of both the honest parties by the end of the first round, protocol GOD is correct.*

Proof: In this case, P_i and P_j does not communicate at all in the second round of GOD_k preventing P_k to compute an output. In GOD_i and GOD_j, P_j and P_i, respectively send their inputs on clear to each other along with $\mathtt{nOK}$ signal. Both compute y on the inputs x_i, x_j that are exchanged and a default common value for x_k by the end of round 2. In the third round, P_k receives y from the honest parties and the honest parties output y. In this case the unique input of the corrupt party taken for computation is the default commonly-agreed value. □

Lemma 3.28 *Assuming that the corrupt party belongs to the corrupt set of exactly one of the honest parties by the end of the first round, protocol* GOD *is correct.*

Proof: For simplicity $P_k \in \mathcal{C}_i$ at the end of first round. (The proof follows in a similar way when $P_k \in \mathcal{C}_j$.) This implies P_i, as an evaluator, ignores communication from both the garblers in its execution GOD_i and will conclude the second round with $y = \perp$ and $\mathcal{C}_i = P_k$. P_i does not participate in GOD_k as a garbler making sure P_k cannot compute an output by the end of second round. In GOD_j, P_i sends x_i on clear to P_j with $\mathtt{nOK}$ signal which implies evaluation of the GC created by P_k is ruled out. Now based on whether P_k commits to any input or not, P_j computes the output in the following way. If $\mathtt{nOK}$ signal is sent along with its input x_k, then P_j computes $y = y_i = y_k$ using its own input x_j and the inputs sent by P_i and P_k. If P_k sends $\mathtt{OK}$ with its encoded input which verifies correctly with respect to the committed encoded information, P_j obtains $y = y_i$ upon GC (C_i) evaluation. In the case when P_k does not commit to any input either on clear or in encoded form (namely, the encoded input does not verify against the committed encoded input), P_j must have identified P_k to be corrupt and computes y using its own input x_j, the input sent by P_i and using a default value for x_k. The third round is finally used by P_i and P_k to obtain the output of P_j and correctness follows. The unique input of P_k is taken as the one that it sends either on clear or in encoded form to P_j in the former case and a default value in the latter. □

Lemma 3.29 *Assuming that the corrupt party does not belong to the corrupt set of both the honest parties by the end of the first round, protocol* GOD *is correct.*

Proof: In this case, P_k must have 'committed' (Definition 3.1) to his input (else would be identified by atleast one of the honest parties at end of Round 1) and obtained output y based on its committed input during GOD_k. Further, P_k is not detected yet by the end of first round, implies that it has played the role of the garblers in GOD_i and GOD_j honestly in the first round. In this case, we prove that no matter how P_k behaves in the second round, the honest parties will obtain y based on their inputs and P_k's committed input. We present the argument

for honest P_i. Similar argument holds for P_j. Based on the observation that P_i must have attempted to evaluate C_k since P_j must have sent $\mathtt{OK}$ signal in GOD_i, we consider the following cases:

- P_i is unsuccessful in evaluating the circuit C_k of garbler P_k in GOD_i. This implies P_k has given inconsistent encoded input for its circuit to P_i. So P_i concludes the second round with $y = \perp$ and $\mathcal{C}_i = P_k$.

- P_i is successful in evaluating the circuit C_k of garbler P_k in GOD_i. By Lemma 3.26, P_k must have given encoded input corresponding to its committed input x_k for C_k. This implies the output obtained via C_k (i.e y_k) is the desired y in this case. Now we have two cases based on whether P_k approves the garbled circuit constructed by P_j or not. In each case we show that, P_i outputs the desired y by the end of second round itself. If P_k disapproves, then $y_j = \perp$ and P_i outputs the value $y = y_k$ obtained via the GC C_k as per the specification of GOD_i. Otherwise, P_i evaluates both circuits, namely C_j and C_k. If the outputs are the same, then the guarantee provided by Lemma 3.26 implies P_i outputs the desired y. Else if P_i has got conflicting outputs ($y_j \neq y_k$), then it gets access to the key $\mathsf{Y}_j^{y_j} \oplus \mathsf{Y}_k^{y_k}$ and uses it to decrypt at least one of the ciphertexts $\{\mathsf{ct}_j^{y_j}, \mathsf{ct}_k^{y_j}\}$ generated by P_j and P_k. If the decryption of only the honest party P_j's ciphertext succeeds, then P_i obtains $(\mathrm{o}_{jk}, \mathrm{o}_{kj})$, retrieves his missing shares x_{jk}, x_{kj} and computes y using x_i, $x_j = x_{ji} \oplus x_{jk}$ and $x_k = x_{ki} \oplus x_{kj}$ where P_i and P_j receives x_{ki} and x_{kj} respectively from P_k in GOD_k. Even if corrupt P_k's ciphertext is decrypted successfully, the y computed is still as above due the fact that P_k cannot open a different value for x_{jk}, x_{kj} due to the binding property of NICOM. P_i retains this output in the third round.

In the former case, if both P_i and P_j outputs $\perp$ in the end of second round, then the third round is used by P_i and P_j to exchange their inputs and the shares of x_k that they possess. By the end of third round P_i (and P_j as well) outputs the desired y. If P_j was successful in computing y in GOD_j, then P_j sends the output directly in third round which P_i takes as the output. In the latter case, P_i retains his output in the third round. □

Theorem 3.5 *Protocol* GOD *is correct.*

Proof: The proof follows from Lemma 3.27,3.28,3.29 as we have considered all the cases exhaustively based on whether the corrupt party P_k is identified by none, exactly one or both the honest parties by the end of first round. □

3.6 Optimizations

In this section, we propose some optimizations to our protocols Fair, UAbort and GOD that will reduce their communication. To reduce total communication, the transmission of garbled circuits should be kept minimal since they constitute the dominant part of communication. We note that the protocols already ensure that each distinct GC is communicated only once to the evaluator, namely when a garbler sends the opening of the co-garbler's circuit. Next, a proposed optimization to reduce communication is that H of the GC could be committed rather than the GC itself, where H denotes a collision-resistant hash function. Infact since broadcast communication is considered more expensive than private communication, corresponding to broadcast of a message, say m, let $\mathsf{H}(m)$ be the message broadcast by the sender while m is sent privately over pairwise channels. The same trick can be applied on the redundant common messages sent over pairwise channels as well i.e if both P_1, P_2 are supposed to send m to P_3, then have P_1 send m and P_2 send $\mathsf{H}(m)$. P_3 can locally compute the hash of the message which would suffice to verify if P_1 and P_2 agree on a common m. The above techniques reduce total communication and makes the broadcast communication complexity of the protocol independent of the circuit size. Lastly, an optimization with respect to protocol Fair is that the inputs to the subprotocol Cert_i can be modified to hash of the relevant inputs instead, reducing considerably the size of the equality-checking circuit in Cert_i.

3.7 Security Proofs

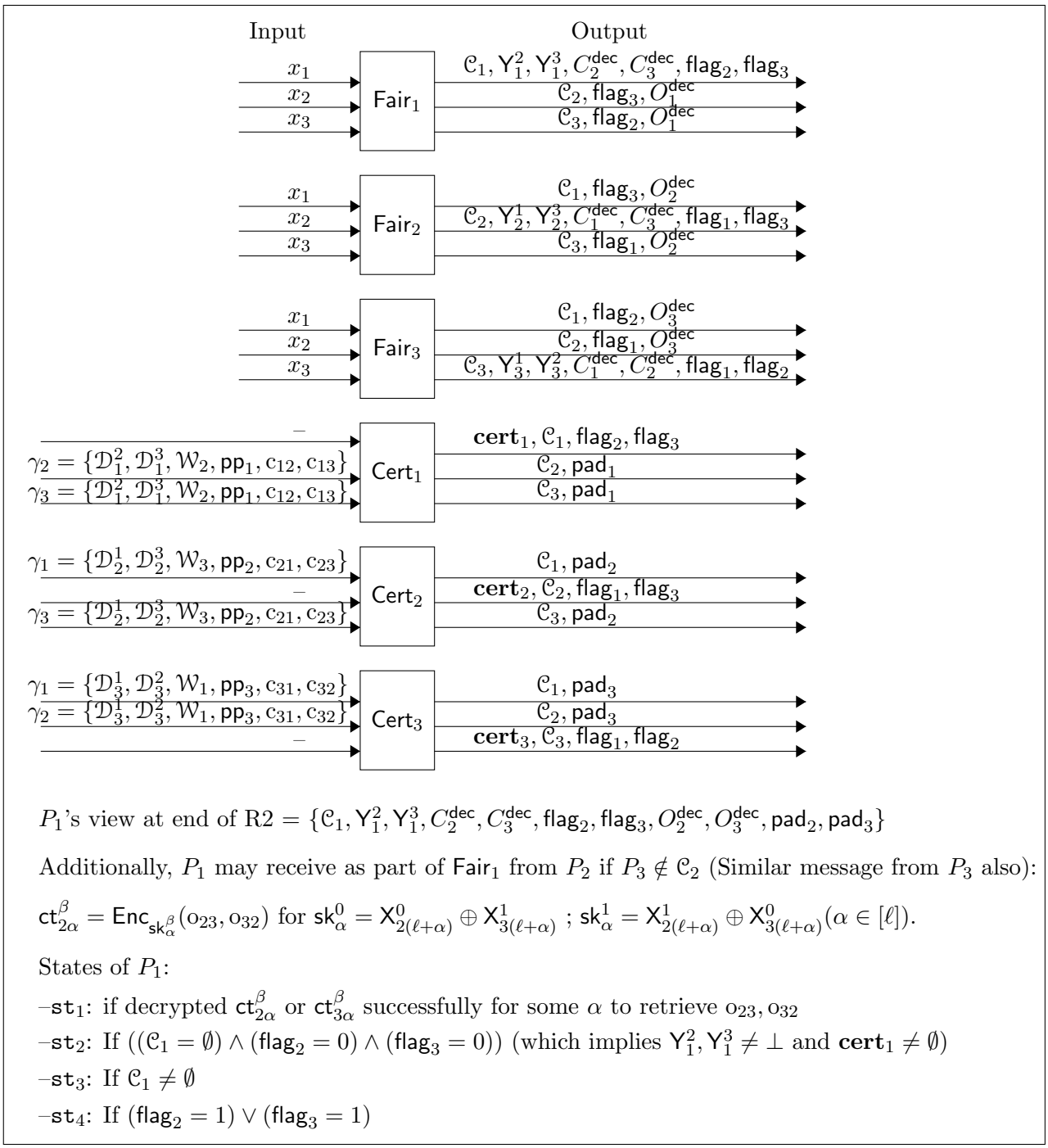
3.7.1 Round Optimal 3PC with fairness

3.7.1.1 Schematic Diagram

We present the schematic diagram of the 3-round Fair protocol in Figures 3.8 - 3.9.

3.7.1.2 Formal Proof of Security for Protocol Fair

In this section, we present the proof of security of Fair relative to the ideal functionality for fn (Figure 2.3). For better clarity, we assume without loss of generality that P_1 is corrupt (denoted as P_1^*) and describe the simulator $\mathcal{S}_{\mathsf{Fair}}$. Since the roles of the parties are symmetric in Fair, similar proof would hold in case of corrupt P_2, P_3 as well. The simulator plays the role of the honest parties P_2, P_3 and simulates each step of the protocol Fair. Recall that during the first two rounds of Fair, the two round protocols Fair_i ($i \in [3]$) and Cert_i ($i \in [3]$) run in parallel. We divide the description of $\mathcal{S}_{\mathsf{Fair}}$ as follows: We describe $\mathcal{S}_{\mathsf{Fair}}$ during $\mathsf{Fair}_1, \mathsf{Cert}_1$ where corrupt P_1^* is the evaluator and during $\mathsf{Fair}_2, \mathsf{Cert}_2$ when corrupt P_1^* acts as a garbler.



P_1's view at end of R2 $= \{\mathcal{C}_1, \mathsf{Y}_1^2, \mathsf{Y}_1^3, C_2^{\mathsf{dec}}, C_3^{\mathsf{dec}}, \mathsf{flag}_2, \mathsf{flag}_3, O_2^{\mathsf{dec}}, O_3^{\mathsf{dec}}, \mathsf{pad}_2, \mathsf{pad}_3\}$

Additionally, P_1 may receive as part of Fair_1 from P_2 if $P_3 \notin \mathcal{C}_2$ (Similar message from P_3 also):

$\mathsf{ct}_{2\alpha}^{\beta} = \mathsf{Enc}_{\mathsf{sk}_\alpha^\beta}(\mathrm{o}_{23}, \mathrm{o}_{32})$ for $\mathsf{sk}_\alpha^0 = \mathsf{X}_{2(\ell+\alpha)}^0 \oplus \mathsf{X}_{3(\ell+\alpha)}^1$; $\mathsf{sk}_\alpha^1 = \mathsf{X}_{2(\ell+\alpha)}^1 \oplus \mathsf{X}_{3(\ell+\alpha)}^0 (\alpha \in [\ell])$.

States of P_1:

–$\mathtt{st}_1$: if decrypted $\mathsf{ct}_{2\alpha}^{\beta}$ or $\mathsf{ct}_{3\alpha}^{\beta}$ successfully for some α to retrieve $\mathrm{o}_{23}, \mathrm{o}_{32}$

–$\mathtt{st}_2$: If $((\mathcal{C}_1 = \emptyset) \wedge (\mathsf{flag}_2 = 0) \wedge (\mathsf{flag}_3 = 0))$ (which implies $\mathsf{Y}_1^2, \mathsf{Y}_1^3 \neq \perp$ and $\mathbf{cert}_1 \neq \emptyset$)

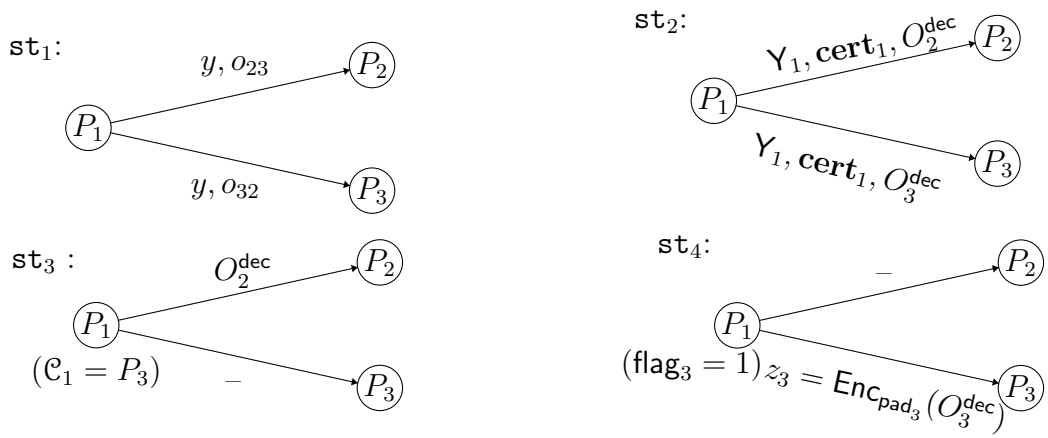
–$\mathtt{st}_3$: If $\mathcal{C}_1 \neq \emptyset$

–$\mathtt{st}_4$: If $(\mathsf{flag}_2 = 1) \vee (\mathsf{flag}_3 = 1)$

Figure 3.8: Schematic Diagram of Fair protocol (Round 1 and 2)

The steps corresponding to Fair_3, and Cert_3 would follow symmetrically from that described corresponding to $\mathsf{Fair}_2, \mathsf{Cert}_2$. Finally, we describe the steps corresponding to the third round. The simulator $\mathcal{S}_{\mathsf{Fair}}$ appears in Figure 3.10 with **R1/R2/R3** indicating simulation for round 1, 2 and 3 respectively and **f/c/F** denoting the steps corresponding to subprotocol $\mathsf{Fair}_i, \mathsf{Cert}_i, \mathsf{Fair}$ respectively.

Figure 3.9: Schematic Diagram of Protocol Fair (Round 3 wrt P_1)



Output Computation:

– $\mathtt{st}_1$: Output y

– $\{\mathtt{st}_2, \mathtt{st}_3, \mathtt{st}_4\}$: Output y if y received from P_2 or P_3 with valid o_{13} or o_{12} respectively

– $\mathtt{st}_2$: Retrieve O_1^{dec} either directly or using z_1 (with $\mathbf{cert}_1$ as the key) from P_2.
If $\mathsf{d} \leftarrow \mathsf{eOpen}(\mathsf{epp}_3, C_3^{\mathsf{dec}}, O_1^{\mathsf{dec}})$ is not $\perp$, compute $y \leftarrow \mathsf{De}(\mathsf{Y}_1^3, \mathsf{d})$. (Similar steps wrt P_3)

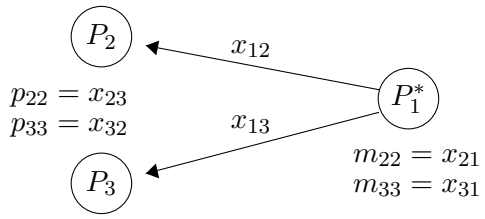
– $\mathtt{st}_3$: (Let $\mathcal{C}_1 = P_3$) If received Y_2 from P_2, output $y \leftarrow \mathsf{De}(\mathsf{Y}_2^1, \mathsf{d})$. Else output $\perp$

– $\mathtt{st}_4$: (Let $\mathsf{flag}_3 = 1$) If $\mathbf{cert}_3 = \mathsf{pad}_3$ received from P_3, update $\mathcal{C}_1 = P_2$ and go to $\mathtt{st}_3$. Else output $\perp$.

When simulating Fair_1, the simulator does not have access to the inputs of the honest parties. Further, it does not know if and what P_1 commits as its input in Round 1, when simulating and sending the commitments for GC and encoding information in parallel in Round 1. Nor does it know if all the parties will get the output (relative to corrupt P_1's committed input from Round 1) or not, when it opens the encoded input and GC in Round 2. The decision comes from P_1's behaviour in Round 2. A privacy simulator $\mathcal{S}_{\mathsf{prv}}$ cannot be invoked for emulating Round 2 message, as $\mathcal{F}_{\mathsf{fair}}$ cannot be invoked yet and so y is not available. Instead oblivious simulator $\mathcal{S}_{\mathsf{obv}}$ is invoked that works without y. Later if and when $\mathcal{F}_{\mathsf{fair}}$ is invoked and y is known, $\mathcal{S}_{\mathsf{prv}}$ is invoked which simply returns the decoding information that makes the fake GC returned by $\mathcal{S}_{\mathsf{obv}}$ output y.

Simulator $\mathcal{S}_{\mathsf{Fair}}$

$\mathcal{S}_{\mathsf{Fair}}$ **during** $\mathsf{Fair}_1, \mathsf{Cert}_1, \mathsf{Fair}$



R1 f: Sample $\mathsf{epp}_2, \mathsf{epp}_3$ for eCom, having trapdoor t_2, t_3. Choose $m_{22} = x_{21}$ (sent during Fair_2), $m_{33} = x_{31}$ (sent during Fair_3), m_{23}, m_{32} at random. On behalf of P_i ($i \in \{2,3\}$) compute $(\mathsf{C}_i, e_i, d_i) \leftarrow \mathsf{Gb}(1^\kappa, C)$ using uniform randomness. Send $\mathcal{D}_i = (\mathsf{epp}_i, \mathsf{pp}_i, \mathrm{c}_i, \{\mathrm{c}^b_{i\alpha}\}_{\alpha\in[4\ell],b\in\{0,1\}}, \mathrm{c}^{\mathsf{dec}}_i)$ to P_1^* where c_i, $\{\mathrm{c}^{m^\alpha_{i2}}_{i\alpha}, \mathrm{c}^{m^\alpha_{i3}}_{i(\ell+\alpha)}, \mathrm{c}^0_{i(2\ell+\alpha)}, \mathrm{c}^1_{i(2\ell+\alpha)}, \mathrm{c}^0_{i(3\ell+\alpha)}, \mathrm{c}^1_{i(3\ell+\alpha)}\}_{\alpha\in[\ell]}$ be computed as per the protocol. Let $\mathrm{c}^{\mathsf{dec}}_i$ and remaining $\{\mathrm{c}^b_{i\alpha}\}$ commit to dummy values. (For Naor-based eNICOM, set c_i, $\mathrm{c}^{\mathsf{dec}}_i$ to the specific commitment supporting equivocation)

R1 c: As per the protocol, compute and send $\mathcal{W}_1$ to P_1^* on behalf of P_3.

R1 f: Receive x_{12}, x_{13} from P_1^* on behalf of P_2, P_3 respectively.

R1 F: Receive $(\mathsf{pp}_1, \mathrm{c}_{12}, \mathrm{c}_{13}, \mathrm{o}_{12})$ on behalf of P_2 and $(\mathsf{pp}_1, \mathrm{c}_{12}, \mathrm{c}_{13}, \mathrm{o}_{13})$ on behalf of P_3 from P_1^*. Set $\mathcal{C}_i = \{P_1\}$, $i \in \{2,3\}$ if $\mathsf{Open}(\mathsf{pp}_i, \mathrm{c}_{1i}, \mathrm{o}_{1i}) \neq x_{1i}$.

R2 f: If $P_1 \notin \mathcal{C}_2, \mathcal{C}_3$, run $\mathsf{C}'_i \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C, \mathsf{X}_i = \{e^{m^\alpha_{i2}}_{i\alpha}, e^{m^\alpha_{i3}}_{i(\ell+\alpha)}, e^{x^\alpha_{12}}_{i(2\ell+\alpha)}, e^{x^\alpha_{13}}_{i(3\ell+\alpha)}\}_{\alpha\in[\ell]})$. Using trapdoor t_i, compute $\mathrm{o}_i = \mathsf{Equiv}(\mathrm{c}_i, \mathsf{C}'_i, t_i)$. Send $\mathtt{OK}$ message on behalf of P_2, P_3 as per protocol using computed $\mathrm{o}_2, \mathrm{o}_3$.

R2 f: Else if $P_1 \notin \mathcal{C}_i$, then act on behalf of P_i as per the protocol (For Naor-based eNICOM equivocate c_i to C_i using t_i.)

R2 F: Set $\mathsf{flag}_1 = 1$ on behalf of both P_2, P_3 if either $P_1 \in \mathcal{C}_2$ or $P_1 \in \mathcal{C}_3$ or $\{\mathsf{pp}_1, \mathrm{c}_{12}, \mathrm{c}_{13}\}$ received on behalf of P_2, P_3 are not identical.

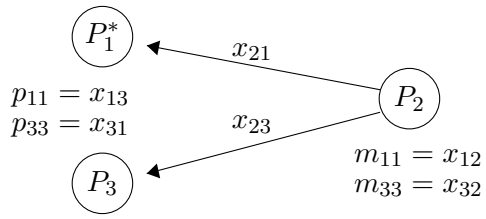
R2 F: Send ciphertext ct on dummy message on behalf of P_i if $P_1 \notin \mathcal{C}_i$ ($i \in \{2,3\}$).

R2 F: If $P_1 \notin \mathcal{C}_i$, $\gamma_i = \{\mathcal{D}^1_1, \mathcal{D}^3_1, \mathcal{W}_2, \mathsf{pp}_1, \mathrm{c}_{12}, \mathrm{c}_{13}\}$ received from P_1 on behalf of P_i ($i \in \{2,3\}$).

R2 c: If $P_1 \notin \mathcal{C}_2$, send $\mathsf{o}_1, \mathcal{W}_1$ (same as computed on behalf of P_3 in Round 1) and (opening of) encoding of γ_2 to P_1 on behalf of P_2 as per the protocol.

R2 c: If $P_1 \notin \mathcal{C}_3$, send (opening of) encoding of γ_3 to P_1 on behalf of P_3.

$\mathcal{S}_{\mathsf{Fair}}$ **during** $\mathsf{Fair}_2, \mathsf{Cert}_2, \mathsf{Fair}$



R1 f: Choose x_{21} at random and send to P_1^* on behalf of P_2.

R1 F: Let $p_{33} = x_{31}$ (sent during Fair_3).

R1 F: Sample pp_2 to compute $(\mathrm{c}_{21}, \mathrm{o}_{21}) \leftarrow \mathsf{Com}(\mathsf{pp}_2, x_{21})$. Send $\{\mathsf{pp}_2, \mathrm{c}_{21}, \mathrm{c}_{23}, \mathrm{o}_{21}\}$ to P_1^* where c_{23} is dummy commitment.

R1 f: Compute and send $\mathcal{D}_3$ and the information associated with $\mathcal{D}_3$ to P_1^* on behalf of P_3 according to the protocol.

R1 f: Receive $\mathcal{D}_1$ and associated information privately from P_1^* on behalf of P_3. Do all the verifications as an honest P_3 would perform for P_1 and update $\mathcal{C}_3$.

R1 F: Add P_1 to $\mathcal{C}_3$ if $p_{11} \neq x_{13}$ (received in Fair_1).

R1 c: Receive $\mathcal{W}_2$ from P_1^* on behalf of P_2.

R1 c: Receive $(s_2, \mathcal{W}_2)$ from P_1^* on behalf of P_3. Do all the verifications and update $\mathcal{C}_3$ as per the protocol.

R2 f: Send $\mathcal{D}_3$ (as computed on behalf of P_3) to P_1^* on behalf of P_2.

R2 f: Set $\mathsf{flag}_1 = 1$ and $\mathsf{Y}_2^1 = \perp$ on behalf of P_2 if $P_1 \in \mathcal{C}_3$ (equivalent to receiving $\mathtt{nOK}$ from P_3) or $P_1 \notin \mathcal{C}_3$ but P_1 sends something other than $\mathcal{D}_1$ (known to P_2 as simulator runs on behalf of P_3)

R2 f: Set $\mathsf{flag}_3 = 1$ and $\mathsf{Y}_2^3 = \perp$ on behalf of P_2 if P_1 sends $\mathtt{nOK}$ or sends $\mathtt{OK}$ with something other than $\mathcal{D}_3$ (known to P_2 as simulator runs on behalf of P_3).

R2 f: If $\mathsf{flag}_1 = \mathsf{flag}_3 = 0$ wrt P_2, set $\mathcal{C}_2 = P_1$ if any of the decommitments (corresponding to C_3 or encoded inputs corresponding to $\mathsf{C}_1, \mathsf{C}_3$) sent by P_1^* opens to something other than what was originally committed (known on behalf of P_3).

R2 F: Set $\mathcal{C}_2 = P_1$ if $m_{11} \neq x_{12}$

R2 F: Send $\{\mathsf{pp}_2, \mathrm{c}_{21}, \mathrm{c}_{23}\}$ (as sent on behalf of P_2) to P_1^* on behalf of P_3. Set $\mathsf{flag}_2 = 1$ wrt P_3 if nothing / other than $\{\mathsf{pp}_2, \mathrm{c}_{21}, \mathrm{c}_{23}\}$ received from P_1^*.

R2 c: Set $\mathbf{cert}_2 = \perp$ and $\mathsf{flag}_1 = 1$ on behalf of P_2 if either $P_1 \in \mathcal{C}_3$ (equivalent to receiving $\mathtt{nOK}$ from P_3) or $\mathcal{C}_3 = \emptyset$ but P_1^* sends $\mathcal{W}_2$ different from one received on behalf of P_3 in Round 1.

R2 F: Else, set $\mathcal{C}_2 = P_1$ if P_1 sends opening of encoded input (known on behalf of P_3) that opens to anything other than the encoding of value $\gamma_1 = \{\mathcal{D}_2^1, \mathcal{D}_2^3, \mathcal{W}_3, (\mathsf{pp}_2, c_{21}, c_{23})\}$ sent on behalf of P_2 during $\mathsf{Fair}_1, \mathsf{Fair}_3, \mathsf{Cert}_3$ and Fair respectively.

$\mathcal{S}_{\mathsf{Fair}}$ during Round 3:

R3 Suppose $\mathcal{C}_2 = \emptyset, \mathsf{flag}_1 = \mathsf{flag}_3 = 0$ wrt P_2: If P_1 sends encoded inputs corresponding to mismatched input bit across $\mathsf{C}_1, \mathsf{C}_3$ during Fair_2 (known on behalf of P_3), mark P_2 as being in $\mathtt{st}_1$. Invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{sid}, \mathsf{Input}, x_1)$ to obtain y where $x_1 = x_{12} \oplus x_{13}$. Send (y, o_{13}) to P_1 on behalf of P_2. Similar steps are executed on behalf of P_3 if $\mathcal{C}_3 = \emptyset, \mathsf{flag}_1 = \mathsf{flag}_2 = 0$.

R3 For every input bit of P_3, choose s bits uniformly at random, say $b_1, \ldots b_s$. Using key based on P_3's consistent input b_α ($\alpha \in [s]$) used in $\mathsf{C}_1, \mathsf{C}_3$ during Fair_2, try to decrypt ciphertext $\mathsf{ct}_{1\alpha}^{\beta}$ for ($\beta \in \{0,1\}$) received from P_1 in Round 2. If the decryption is successful and the openings retrieved are same as $(\mathsf{o}_{13}, \mathsf{o}_{31})$, mark P_2 as being in $\mathtt{st}_1$ and do the following: invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{sid}, \mathsf{Input}, x_1)$ to obtain y where $x_1 = x_{12} \oplus x_{13}$. Send (y, o_{13}) to P_1 on behalf of P_2. Similar steps are executed by the simulator on behalf of P_3

R3 If P_2 or P_3 is in $\mathtt{st}_2$, let $x_1 = x_{12} \oplus x_{13}$. Invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{sid}, \mathsf{Input}, x_1)$ to obtain y.

R3 If P_2 in $\mathtt{st}_2$, to retrieve decoding information $\mathsf{o}_3^{\mathsf{dec}}$: Run $(\mathsf{C}_3, \mathsf{d}_3^1) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \mathsf{X} = \{e_{3\alpha}^{m_{i2}^\alpha}, e_{3(\ell+\alpha)}^{m_{i3}^\alpha}, e_{3(2\ell+\alpha)}^{x_{12}^\alpha}, e_{3(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha\in[\ell]})$. Equivocate the commitment on decoding information in Fair_1 ($\mathsf{c}_3^{\mathsf{dec}}$) to get $\mathsf{o}_3^{\mathsf{dec}} = \mathsf{Equiv}(\mathsf{c}_3^{\mathsf{dec}}, d_3^1, t_3)$. Send $(\mathsf{Y}_2, \mathbf{cert}_2, \mathsf{o}_3^{\mathsf{dec}})$ to P_1^* on behalf of P_2. Here $\mathbf{cert}_2$ is set as encoding of 1 on output wire of C_2 during Cert_2 and Y_2 is the encoding corresponding to output y of $\mathsf{C}_1, \mathsf{C}_3$ during Fair_2; both of which are known as simulator acts on behalf of P_3.

R3 Similar steps as above if P_3 is in $\mathtt{st}_2$.

R3 Invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{sid}, \mathsf{Input}, \mathtt{abort})$ if neither P_2 nor P_3 belong to $\{\mathtt{st}_1, \mathtt{st}_2\}$.

R3 Send dummy ciphertext z_1 to P_1^* on behalf of P_i, $i \in \{2,3\}$ if P_i in $\mathtt{st}_4$ with $\mathsf{flag}_1 = 1$.

Figure 3.10: Description of $\mathcal{S}_{\mathsf{Fair}}$

We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}_{\mathsf{Fair}}} \overset{c}{\approx} \text{REAL}_{\mathsf{Fair}, \mathcal{A}}$, when $\mathcal{A}$ corrupts P_1. The views are shown to be indistinguishable via a series of intermediate hybrids.

– HYB_0: Same as $\text{REAL}_{\mathsf{Fair}, \mathcal{A}}$.

– HYB_1: Same as HYB_0, except that P_2, P_3 in Fair_1 use uniform randomness rather than

pseudo-randomness for the garbled circuit construction.

– HYB_2: Same as HYB_1, except that some of the commitments of encoded inputs which will not be sent to P_1 during Fair_1 are replaced with commitments on dummy values. Specifically, these are corresponding to indices not equal to $m_{22}, m_{23}, x_{12}, x_{13}$ for C_2 and not equal to $m_{32}, m_{33}, x_{12}, x_{13}$ for C_3.

– HYB_3 : Same as HYB_2, except the following:

- $\text{HYB}_{3.1}$: When the execution results in P_1 evaluating GCs during Fair_1 but results in $\mathtt{abort}$, C_2 is created as $\mathsf{C}'_2 \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C, \mathsf{X}_2 = \{e^{m^\alpha_{22}}_{2\alpha}, e^{m^\alpha_{23}}_{2(\ell+\alpha)}, e^{x^\alpha_{12}}_{2(2\ell+\alpha)}, e^{x^\alpha_{13}}_{2(3\ell+\alpha)}\}_{\alpha\in[\ell]})$. The commitment c_2 is later equivocated to C'_2 using o_2 computed via $\mathrm{o}_2 \leftarrow \mathsf{Equiv}(\mathrm{c}_2, \mathsf{C}'_2, t_2)$. The commitment to the decoding information is created for a dummy value. Since the encoding information are committed in round 1 using committing commitments that cannot be equivocated, we invoke $\mathcal{S}_{\mathsf{obv}}$ using an X that corresponds to the correct shares of P_1 and it returns a fake GC (consistent with the labels in X) such that indistinguishability holds. We note that most of the known garbling schemes based on Yao and optimizations [182, 183, 141] have simulators that comply with the above.
- $\text{HYB}_{3.2}$: When the execution results in P_1 evaluating GCs during Fair_1 and output y, the GC is created as $(\mathsf{C}'_2, d_2) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \mathsf{X}_2 = \{e^{m^\alpha_{22}}_{2\alpha}, e^{m^\alpha_{23}}_{2(\ell+\alpha)}, e^{x^\alpha_{12}}_{2(2\ell+\alpha)}, e^{x^\alpha_{13}}_{2(3\ell+\alpha)}\}_{\alpha\in[\ell]})$. The commitment c_2 is later equivocated to C'_2 using o_2 computed via $\mathrm{o}_2 \leftarrow \mathsf{Equiv}(\mathrm{c}_2, \mathsf{C}'_2, t_2)$. The commitment $\mathrm{c}^{\mathsf{dec}}_2$ to the decoding information is created for a dummy value and later equivocated to d_2 using $\mathrm{o}_{\mathsf{d}_2}$ computed via $\mathrm{o}_{\mathsf{d}_2} \leftarrow \mathsf{Equiv}(\mathrm{c}^{\mathsf{dec}}_2, \mathsf{d}_2, t_2)$. The set of ciphertexts ct and z_1 (if) generated use d_2 .

– HYB_4 : Same as HYB_2, except the following:

- $\text{HYB}_{4.1}$: When the execution results in P_1 evaluating GCs during Fair_1 but results in $\mathtt{abort}$, C_3 is created as $\mathsf{C}'_3 \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C, \mathsf{X}_3 = \{e^{m^\alpha_{32}}_{3\alpha}, e^{m^\alpha_{33}}_{3(\ell+\alpha)}, e^{x^\alpha_{12}}_{3(2\ell+\alpha)}, e^{x^\alpha_{13}}_{3(3\ell+\alpha)}\}_{\alpha\in[\ell]})$. The commitment c_3 is later equivocated to C'_3 using o_3 computed via $\mathrm{o}_3 \leftarrow \mathsf{Equiv}(\mathrm{c}_3, \mathsf{C}'_3, t_3)$. The commitment to the decoding information is created for a dummy value.
- $\text{HYB}_{4.2}$: When the execution results in P_1 evaluating GCs during Fair_1 and output y, the GC is created as $(\mathsf{C}'_3, d_3) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \mathsf{X}_3 = \{e^{m^\alpha_{32}}_{3\alpha}, e^{m^\alpha_{33}}_{3(\ell+\alpha)}, e^{x^\alpha_{12}}_{3(2\ell+\alpha)}, e^{x^\alpha_{13}}_{3(3\ell+\alpha)}\}_{\alpha\in[\ell]})$. The commitment c_3 is later equivocated to C'_3 using o_3 computed via $\mathrm{o}_3 \leftarrow \mathsf{Equiv}(\mathrm{c}_3, \mathsf{C}'_3, t_3)$. The commitment $\mathrm{c}^{\mathsf{dec}}_3$ to the decoding information is created for a dummy

value and later equivocated to d_3 using $\mathrm{o}_{\mathsf{d}_3}$ computed via $\mathrm{o}_{\mathsf{d}_3} \leftarrow \mathsf{Equiv}(\mathrm{c}_3^{\mathsf{dec}}, \mathsf{d}_3, t_3)$. The set of ciphertexts ct and z_1 (if) generated uses d_3.

- HYB_5: Same as HYB_4, except that during Fair_2, $\mathcal{C}_2$ is set to P_1 if P_2 receives o_3 that opens to a value other than the originally committed C_3.

- HYB_6: Same as HYB_5, except that during Fair_3, $\mathcal{C}_3$ is set to P_1 if P_3 receives o_2 that opens to a value other than the originally committed C_2.

- HYB_7: Same as HYB_6, except that during Fair_2, $\mathcal{C}_2$ is set to P_1 if P_2 accepts any encoded input not consistent with $\mathsf{C}_1, \mathsf{C}_3$

- HYB_8: Same as HYB_7, except that during Fair_3, $\mathcal{C}_3$ is set to P_1 if P_3 accepts any encoded input not consistent with $\mathsf{C}_1, \mathsf{C}_2$

- HYB_9: Same as HYB_8, except that when the execution does not result in P_1 getting access to the opening of commitment c_{23} (corresponding to x_{23}) sent by P_2 during Fair_2, the commitment is replaced with commitment of dummy value.

- HYB_{10}: Same as HYB_9, except that when the execution does not result in P_1 getting access to the opening of commitment c_{32} (corresponding to x_{32}) sent by P_3 during Fair_3, the commitment is replaced with commitment of dummy value.

- HYB_{11}: Same as HYB_{10}, except that when the execution Fair_1 does not result in P_1 getting encoded inputs corresponding to mismatched input bit across the two garbled circuits corresponding to any garbler, the set of ct is replaced by encryption of a dummy message.

- HYB_{12}: Same as HYB_{11}, except that during Cert_2, P_2 (with $\mathsf{flag}_1 = 0$) adds P_1 to $\mathcal{C}_2$ if (opening of) encoded input sent by P_1 corresponding to C_2 is anything other than the opening of the originally committed encoded information corresponding to value $\gamma = \{\mathcal{D}_2^1, \mathcal{D}_2^3, \mathcal{W}_3, (\mathsf{pp}_2, \mathrm{c}_{21}, \mathrm{c}_{23})\}$ sent by P_2 in Round 1.

- HYB_{13}: Same as HYB_{12}, except that during Cert_3, P_3 (with $\mathsf{flag}_2 = 0$) adds P_1 to $\mathcal{C}_3$ if (opening of) encoded input sent by P_1 corresponding to C_3 is anything other than the opening of the originally committed encoded information corresponding to value $\gamma = \{\mathcal{D}_3^1, \mathcal{D}_3^2, \mathcal{W}_1, (\mathsf{pp}_3, \mathrm{c}_{31}, \mathrm{c}_{32})\}$ sent by P_3 in Round 1.

- HYB_{14}: Same as HYB_{13}, except that during Cert_1, when P_1's evaluation of C_1 does not result in output 1, z_1 (if) sent to P_1 is replaced with encryption of dummy message.

– HYB_{15}: Same as HYB_{14}, except that $\mathsf{Y}_2^1, \mathsf{Y}_2^3$ is computed via $\mathsf{De}(\mathsf{Y}_2^1, d_1) = y$, $\mathsf{De}(\mathsf{Y}_2^3, d_3) = y$, (where d_1, d_3 correspond to decoding information of $\mathsf{C}_1, \mathsf{C}_3$ during Fair_2) rather than $\mathsf{Y}_2^1 = \mathsf{Ev}(\mathsf{C}_1, \mathsf{X})$, $\mathsf{Y}_2^3 = \mathsf{Ev}(\mathsf{C}_3, \mathsf{X})$.

– HYB_{16}: Same as HYB_{15}, except that $\mathsf{Y}_3^1, \mathsf{Y}_3^2$ is computed via $\mathsf{De}(\mathsf{Y}_3^1, d_1) = y$, $\mathsf{De}(\mathsf{Y}_3^2, d_2) = y$ (where d_1, d_2 correspond to decoding information of $\mathsf{C}_1, \mathsf{C}_2$ during Fair_3) rather than $\mathsf{Y}_3^1 = \mathsf{Ev}(\mathsf{C}_1, \mathsf{X})$, $\mathsf{Y}_3^2 = \mathsf{Ev}(\mathsf{C}_2, \mathsf{X})$.

– HYB_{17}: Same as HYB_{16}, except that during Cert_2, if P_2 gets access to $\mathsf{Y}_2 \leftarrow (\mathsf{C}_2, \mathsf{X})$ such that $\mathsf{sDe}(\mathsf{Y}_2) = 1$, $\mathbf{cert}_2 = \mathsf{Y}_2$ is computed via $\mathsf{De}(\mathsf{Y}_2, d_2) = 1$ (where d_2 corresponds to decoding information of C_2 during Cert_2) rather than $\mathsf{Y}_2 = \mathsf{Ev}(\mathsf{C}_2, \mathsf{X})$

– HYB_{18}: Same as HYB_{17}, except that during Cert_3, if P_3 gets access to $\mathsf{Y}_3 \leftarrow (\mathsf{C}_3, \mathsf{X})$ such that $\mathsf{sDe}(\mathsf{Y}_3) = 1$, $\mathbf{cert}_3 = \mathsf{Y}_3$ is computed via $\mathsf{De}(\mathsf{Y}_3, d_3) = 1$ (where d_3 corresponds to decoding information of C_3 during Cert_3) rather than $\mathsf{Y}_3 = \mathsf{Ev}(\mathsf{C}_3, \mathsf{X})$

– HYB_{19}: Same as HYB_{18}, except that P_2 sends (y, o_{13}) to P_1 if decryption of ct sent by P_1 during Fair_2 is successful (and includes openings of x_{13}, x_{31} corresponding to original commitments) using P_3's encoding corresponding to random input.

– HYB_{20}: Same as HYB_{19}, except that P_3 sends (y, o_{12}) to P_1 if decryption of ct sent by P_1 during Fair_3 is successful (and includes openings of x_{12}, x_{21} corresponding to original commitments) using P_2's encoding corresponding to random input.

Since $\text{HYB}_{20} := \text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}_{\mathsf{Fair}}}$, we show that every two consecutive hybrids are computationally indistinguishable which concludes the proof.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that P_2, P_3 in Fair_1 use uniform randomness in HYB_1 rather than pseudorandomness as in HYB_0. The indistinguishability follows via reduction to the security of the PRG G.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is some of the commitments of encoded inputs which will not be sent to P_1 during Fair_1 are replaced with commitments on dummy values. The indistinguishability between the hybrids follows from the hiding property of NICOM.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_{3.1}$: The difference between the hybrids is in the way $(\mathsf{C}_2, \mathsf{X})$ is generated when the execution results in $\mathtt{abort}$. In HYB_2, $(\mathsf{C}_2, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C)$ is run, which gives $(\mathsf{C}_2, \mathsf{En}(x, \mathsf{e}))$. In $\text{HYB}_{3.1}$, it is generated as $\mathsf{C}_2' \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C, \mathsf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{x_{12}^\alpha}, e_{2(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha \in [\ell]})$. The

commitment to the garbled circuit is later equivocated to C_2' using o_2 computed via $\mathrm{o}_2 \leftarrow \mathsf{Equiv}(\mathrm{c}_2, \mathsf{C}_2', t_2)$. Additionally, the commitment to the decoding information is created for a dummy value in $\text{HYB}_{3.1}$. The indistinguishability follows via reduction to the obliviousness of the garbling scheme and the usual hiding property of commitment schemes which is implied by the hiding property of eCom.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_{3.2}$: The difference between the hybrids is in the way $(\mathsf{C}_2, \mathsf{X}, \mathsf{d})$ is generated. In HYB_2, $(\mathsf{C}_2, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C)$ is run, which gives $(\mathsf{C}_2, \mathsf{En}(x, \mathsf{e}), \mathsf{d})$. In $\text{HYB}_{3.2}$, it is generated as $(\mathsf{C}_2', \mathsf{d}_2^1) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \mathsf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{x_{12}^\alpha}, e_{2(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha\in[\ell]})$. The commitment to the garbled circuit is later equivocated to C_2' using o_2 computed via $\mathrm{o}_2 \leftarrow \mathsf{Equiv}(\mathrm{c}_2, \mathsf{C}_2', t_2)$. Additionally, the commitment to the decoding information is created for a dummy value and later equivocated to d_2^1 using $\mathrm{o}_2^{\mathsf{dec}}$ computed via $o_2^{\mathsf{dec}} \leftarrow \mathsf{Equiv}(\mathrm{c}_2^{\mathsf{dec}}, \mathsf{d}_2^1, t_2)$. The indistinguishability follows via reduction to the privacy of the garbling scheme and the hiding property of eCom.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: Similar argument as above with respect to C_3.

$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_5$: The difference between the hybrids is that in HYB_4, P_2 sets $\mathcal{C}_2 = P_1$ if the o_3 sent by P_1 in Fair_2 output $\perp$ while in HYB_5, P_2 sets $\mathcal{C}_2 = P_1$ if o_3 sent by P_1 in Fair_2 opens to any value other than C_3. Since the commitment scheme eCom is binding, in HYB_4, P_1 could have decommitted successfully to a different garbled circuit than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_5 \overset{c}{\approx} \text{HYB}_6$: Similar argument as above with respect to P_3 in Fair_3.

$\text{HYB}_6 \overset{c}{\approx} \text{HYB}_7$: The difference between the hybrids is that in HYB_6, P_2 sets $\mathcal{C}_2 = P_1$ if the encoded inputs sent by P_1 in Fair_2 is inconsistent with $\mathcal{D}_1$, $\mathcal{D}_3$, while in HYB_7 $\mathcal{C}_2$ is set to P_1 if P_2 accepts any encoded input not consistent with $\mathsf{C}_1, \mathsf{C}_3$. It follows from the biding property of NICOM that in HYB_6, P_1 could have sent an encoded input not consistent with $\mathsf{C}_1, \mathsf{C}_3$ but consistent with $\mathcal{D}_1$, $\mathcal{D}_3$, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_7 \overset{c}{\approx} \text{HYB}_8$: Similar argument as above with respect to P_3 in Fair_3.

$\text{HYB}_8 \overset{c}{\approx} \text{HYB}_9$: The difference between the hybrids is that when the execution does not result in P_1 getting access to the opening of commitment c_{23} (corresponding to x_{23}) sent by P_2, c_{23}

corresponds to the actual input share x_{23} in HYB_8 while it corresponds to dummy value in HYB_9. The indistinguishability follows from the hiding property of NICOM.

$\text{HYB}_9 \overset{c}{\approx} \text{HYB}_{10}$: Similar argument as above with respect to commitment c_{32} sent by P_3.

$\text{HYB}_{10} \overset{c}{\approx} \text{HYB}_{11}$: The difference between the hybrids is that when the execution Fair_1 does not result in P_1 getting encoded inputs corresponding to mismatched input bits of any garbler on two garbled circuits, in HYB_{10}, the set of ct is the encryption of a opening of input shares while in HYB_{11}, it is replaced with encryption of dummy message. Assuming the encryption key is unknown to P_1 (holds except with negligible probability due to privacy of garbling scheme), indistinguishability follows from the security of the encryption scheme with special correctness.

$\text{HYB}_{11} \overset{c}{\approx} \text{HYB}_{12}$: The difference between the hybrids is that while in HYB_{11}, during Cert_2, P_2 adds P_1 to $\mathcal{C}_2$ if opening of encoded input sent by P_1 results in $\perp$ or C_2 evaluates to 0 revealing P_1's input being not equal to $\gamma = \{\mathcal{D}_2^1, \mathcal{D}_2^3, \mathcal{W}_3, \mathsf{pp}_2, \mathrm{c}_{21}, \mathrm{c}_{23}\}$; while in HYB_{12} P_1 is added to $\mathcal{C}_2$ if he sends anything other than opening of the originally committed encoded information of C_2 corresponding to value $\gamma = \{\mathcal{D}_2^1, \mathcal{D}_2^3, \mathcal{W}_3, \mathsf{pp}_2, \mathrm{c}_{21}, \mathrm{c}_{23}\}$. The indistinguishability follows from the binding of NICOM and the correctness of the privacy-free garbling scheme (used during Cert_2).

$\text{HYB}_{12} \overset{c}{\approx} \text{HYB}_{13}$: Similar argument as above with respect to P_3 during Cert_3.

$\text{HYB}_{13} \overset{c}{\approx} \text{HYB}_{14}$: The difference between the hybrids is that in HYB_{12}, z_1 is set as encryption of the decoding information of Fair_1 while in HYB_{13}, z_1 is replaced with encryption of a dummy message when P_1's evaluation of C_1 during Cert_1 does not lead to output 1. Assuming the encryption key is unknown to P_1 (holds except with negligible probability due to authenticity of privacy-free garbling scheme used in Cert_1), indistinguishability follows from the security of the encryption scheme.

$\text{HYB}_{14} \overset{c}{\approx} \text{HYB}_{15}$: The difference between the hybrids is that in HYB_{14}, P_2 computes $\mathsf{Y}_2 = (\mathsf{Y}_2^1, \mathsf{Y}_2^3)$ via $\mathsf{Ev}(\mathsf{C}_1, \mathsf{X})$, $\mathsf{Y}_2^3 = \mathsf{Ev}(\mathsf{C}_3, \mathsf{X})$, while in HYB_{15}, $\mathsf{Y}_2^1, \mathsf{Y}_2^3$ is computed such that $\mathsf{De}(\mathsf{Y}_2^1, d_1) = y$, $\mathsf{De}(\mathsf{Y}_2^3, d_3) = y$ (where d_1, d_3 is the decoding information corresponding to $\mathsf{C}_1, \mathsf{C}_3$ during Fair_2). Due to the correctness of the garbling scheme, the equivalence of $\mathsf{Y}_2^1, \mathsf{Y}_2^3$ computed via $\mathsf{Ev}(\mathsf{C}_1, \mathsf{X})$, $\mathsf{Ev}(\mathsf{C}_3, \mathsf{X})$ or such that $\mathsf{De}(\mathsf{Y}_2^1, d_1) = y$, $\mathsf{De}(\mathsf{Y}_2^3, d_3) = y$ holds.

$\text{HYB}_{15} \overset{c}{\approx} \text{HYB}_{16}$: Similar argument as above with respect to Y_3 computed by P_3 during Fair_3.

$\text{HYB}_{16} \overset{c}{\approx} \text{HYB}_{17}$: The difference between the hybrids is that in HYB_{16}, if P_2 obtains $\mathsf{Y}_2 \leftarrow \mathsf{Ev}(\mathsf{C}_2, \mathsf{X})$ such that $\mathsf{sDe}(\mathsf{Y}) = 1$, then P_2 sets $\mathbf{cert}_2 = \mathsf{Y}_2$ while in HYB_{15}, in this case $\mathbf{cert}_2$ is set to Y_2 computed such that $\mathsf{De}(\mathsf{Y}_2, d_2) = 1$ (where d_2 is the decoding information corresponding to C_2 during Cert_2). Due to the correctness of the privacy-free garbling scheme, the equivalence of Y_2 computed via $\mathsf{Ev}(\mathsf{C}_2, \mathsf{X})$ or such that $\mathsf{De}(\mathsf{Y}_2, d_2) = y$ holds.

$\text{HYB}_{17} \overset{c}{\approx} \text{HYB}_{18}$: Similar argument as above with respect to $\mathbf{cert}_3$ computed by P_3 during Cert_3.

$\text{HYB}_{18} \overset{c}{\approx} \text{HYB}_{19}$: The difference between the hybrids is that in HYB_{18}, P_2 sends (y, o_{13}) to P_1 if decryption of ct sent by P_1 during Fair_2 is successful using keys based on P_3's encoding of actual input, whereas in HYB_{19}, P_2 sends (y, o_{13}) to P_1 if decryption of ct sent by P_1 during Fair_2 is successful using keys based on P_3's encoding of random input. The indistinguishability between the hybrids follows from the following claim: Consider single bit input for simplicity. For any two different inputs x and x' of P_3, the difference between the probability that P_2 sends (y, o_{13}) to P_1 when P_3's input is x and when P_3's input is x' is at most 2^{-s+1}. The argument can be divided into three cases (similar to [147]). **(1)** Suppose for some $\alpha \in [s]$, P_1 replaces both ciphertexts $\mathsf{ct}^0_{1\alpha}, \mathsf{ct}^1_{1\alpha}$: one based on consistent input 0 of P_3 and other based on consistent input 1 of P_3 (say, $\mathsf{sk}^0_\alpha = \mathsf{X}^0_{1(s+\alpha)} \oplus \mathsf{X}^0_{3(s+\alpha)}$ and $\mathsf{sk}^1_\alpha = \mathsf{X}^1_{1(s+\alpha)} \oplus \mathsf{X}^1_{3(s+\alpha)}$). In this case, P_2 would be able to decrypt the ciphertext successfully regardless of P_3's input with probability 1 and would send (y, o_{13}) to P_2. **(2)** Suppose P_1 replaces exactly one of the two ciphertexts with consistent input corresponding to $1 \leq j < s$. Since the values assigned (in encoding) by P_3 to any proper subset of the s bits are independent of P_3's actual input, P_2 would be able to decrypt the ciphertext successfully with probability $1 - 2^{-j}$ regardless of the actual value of its original input. **(3)** Suppose P_1 replaces one ciphertext based on consistent input for each of the $\alpha \in [s]$ (say all based on consistent value '1'). Then if x had encoding with any one such value ('1' in the example), the ciphertext would be decrypted successfully with probability 1, whereas decryption would be successful with probability $1 - 2^{-s+1}$ if x' had the other value (in the example, P_2 will be unable to decrypt if $x' = 0$ and the encoding of $x' = 0$ was chosen as $x'_\alpha = 0$ for all $\alpha \in [s]$ (where $x' = \oplus_{\alpha=1}^{s} x_\alpha$) which occurs with probability 2^{-s+1}).

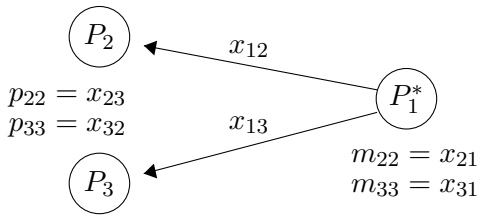
$\text{HYB}_{19} \overset{c}{\approx} \text{HYB}_{20}$: Similar argument as above with respect to ct received by P_3 during Fair_3.

3.7.2 Proof of Security for Protocol UAbort

In this section, we present the proof of security of UAbort relative to the ideal functionality for ua (Figure 2.2). For clarity, we assume without loss of generality that P_1 is corrupt (denoted as P_1^*) and describe the simulator $\mathcal{S}_{\mathsf{UAbort}}$. Since the roles of the parties are symmetric in UAbort, similar proof would hold in case of corrupt P_2, P_3 as well. The simulator plays the role of the honest parties P_2, P_3 and simulates each step of the protocol UAbort. We divide the description of $\mathcal{S}_{\mathsf{UAbort}}$ as follows: We describe $\mathcal{S}_{\mathsf{UAbort}}$ during UAbort_1 where corrupt P_1^* is the evaluator and during UAbort_2 when corrupt P_1^* acts as a garbler. The steps corresponding to UAbort_3, would follow symmetrically from that described corresponding to UAbort_2. The simulator $\mathcal{S}_{\mathsf{UAbort}}$ appears in Figure 3.11 with **R1/R2** indicating simulation for round 1 and 2 respectively and **a/A** denoting the steps corresponding to subprotocol UAbort_i, UAbort respectively. When simulating UAbort_1, the commitments for GC and encoding information need to be simulated and sent in Round 1 itself, while the privacy simulator $\mathcal{S}_{\mathsf{prv}}$ can only be invoked on noting the adversary's behaviour in Round 1 that decides what input it commits and whether it obtains output or $\perp$. Using equivocality of the commitment of GC, we can equivocate the GC as returned by the simulator. But since commitments on the encoding information are committing and the simulator didn't have access to X during simulation of Round 1, the encoded input X returned by $\mathcal{S}_{\mathsf{prv}}$ cannot be explained. So we use a slightly modified version of $\mathcal{S}_{\mathsf{prv}}$ which takes an encoded input (correspond to what will be opened to corrupt P_1) as parameter and returns just the fake GC compatible with it. Yao's privacy simulator can be made to work as above for any encoded input and the indistinguishability will hold with respect to the fake GC and given encoded input.

Simulator $\mathcal{S}_{\mathsf{UAbort}}$

$\mathcal{S}_{\mathsf{UAbort}}$ **during** UAbort_1, UAbort



R1 a: Receive (x_{12}, x_{13}) privately from P_1^* on the behalf of P_2, P_3.

R1 A: Receive $(\mathsf{pp}_1, \mathrm{c}_{12}, \mathrm{c}_{13})$ via broadcast and $\mathrm{o}_{12}, \mathrm{o}_{13}$ privately from P_1^* on behalf of P_2, P_3. Set $\mathsf{flag}_1 = 1$ on behalf of P_i if $\mathsf{Open}(\mathrm{c}_{1i}, \mathrm{o}_{1i}) \neq x_{1i}$ for $i \in \{2, 3\}$.

R1 a: Sample $\mathsf{epp}_2, \mathsf{epp}_3$ for eCom, having trapdoor t_2, t_3. Choose $m_{22} = x_{21}$ (sent during UAbort_2), $m_{33} = x_{31}$ (sent during UAbort_3), m_{23}, m_{32}, r_2, r_3 at random. On behalf of P_i ($i \in \{2,3\}$) compute $(\mathsf{C}_i, e_i, d_i) \leftarrow \mathsf{Gb}(1^\kappa, C)$ using uniform randomness. Broadcast $\mathcal{D}_i = (\mathsf{pp}_i, \mathsf{epp}_i, \mathrm{c}_i, \{\mathrm{c}^b_{i\alpha}\}_{\alpha\in[6\ell],b\in\{0,1\}})$ where $\mathrm{c}_i, \{\mathrm{c}^{m^\alpha_{i2}}_{i\alpha}, \mathrm{c}^{m^\alpha_{i3}}_{i(\ell+\alpha)}, \mathrm{c}^{r^\alpha_2}_{i(2\ell+\alpha)}, \mathrm{c}^{r^\alpha_3}_{i(3\ell+\alpha)}, \mathrm{c}^0_{i(4\ell+\alpha)}, \mathrm{c}^1_{i(4\ell+\alpha)}, \mathrm{c}^0_{i(5\ell+\alpha)}, \mathrm{c}^1_{i(5\ell+\alpha)}\}_{\alpha\in[\ell}$ be computed as per the protocol. (If Naor-based eNICOM is used, c_i should be set to the specific commitment that suppports equivocation as per epp_i.) Let the remaining $\{\mathrm{c}^b_{i\alpha}\}$ commit to dummy values.

R1 a: Send $(\{\mathrm{o}^{m^\alpha_{22}}_{2\alpha}, \mathrm{o}^{r^\alpha_2}_{2(2\ell+\alpha)}\}_{\alpha\in[\ell]}, m_{22}, r_2)$ to P_1^* on behalf of P_2. Send $(\{\mathrm{o}^{m^\alpha_{33}}_{3(\ell+\alpha)}, \mathrm{o}^{r^\alpha_3}_{3(3\ell+\alpha)}\}_{\alpha\in[\ell]}, m_{33}$, $r_3)$ privately to P_1^* on behalf of P_3.

R2 a: If $\mathsf{flag}_1 = 1$ for either P_2 or P_3, invoke $\mathcal{F}_{\mathsf{ua}}$ with $(\mathsf{sid}, \mathsf{Input}, \mathtt{abort})$ on behalf of P_1^*.

R2 a: Broadcast $\mathtt{abort}$ on behalf of P_i ($i \in \{2,3\}$) if $\mathsf{flag}_1 = 1$ on behalf of P_i.

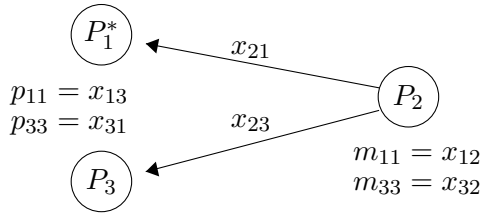
R2 a: If $\mathsf{flag}_1 = 0$ wrt P_i for exactly one $i \in \{2,3\}$, then act on behalf of P_i as per the protocol opening the garbled circuit (equivocate c_i to C_i in case of Naor-based eNICOM) and encoded input as per m_{23} or m_{32} accordingly chosen as above. Broadcast $\mathcal{W}_i$ using $z_i = r_i \oplus x_{1i}$ as per the protocol.

R2 a: If $\mathsf{flag}_1 = 0$ for both P_2 and P_3, invoke $\mathcal{F}_{\mathsf{ua}}$ with $(\mathsf{sid}, \mathsf{Input}, x_1)$ on behalf of P_1^* to obtain output y, where $x_1 = x_{12} \oplus x_{13}$. Let $z_2 = r_2 \oplus x_{12}$ and $z_3 = r_3 \oplus x_{13}$. For $(i \in \{2,3\})$, run $(\mathsf{C}'_i, \mathsf{X}_i, d_i) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \{e^{m^\alpha_{i2}}_{i\alpha}, e^{m^\alpha_{i3}}_{i(\ell+\alpha)}, e^{r^\alpha_2}_{i(2\ell+\alpha)}, e^{r^\alpha_3}_{i(3\ell+\alpha)} e^{z^\alpha_2}_{i(4\ell+\alpha)}, e^{z^\alpha_3}_{i(5\ell+\alpha)}\}_{\alpha\in[\ell]})$. Using trapdoor t_i, compute $\mathrm{o}_i = \mathsf{Equiv}(\mathrm{c}_i, \mathsf{C}'_i, t_i)$. Send $\mathtt{OK}$ message privately to P_1^* on behalf of P_2, P_3 as per the protocol using computed $\mathrm{o}_2, \mathrm{o}_3$. Broadcast $\mathcal{W}_2, \mathcal{W}_3$ on behalf of P_2, P_3 as per protocol.

R2 A: If $\mathsf{flag}_1 \neq 1$ wrt P_i, send set of ct on behalf of P_i ($i \in \{2,3\}$) using a dummy message.

R2 a: Set $\mathsf{flag}_1 = 1$ on behalf of both P_2, P_3 if either (a) $\mathtt{abort}$ was sent or received via broadcast in Round 2 (b) P_1 broadcasts $z_2 \neq x_{12} \oplus r_2$ or $z_3 \neq x_{13} \oplus r_3$

$\mathcal{S}_{\mathsf{UAbort}}$ **during** $\mathsf{UAbort}_2, \mathsf{UAbort}$



R1 A: Set $p_{33} = x_{31}$ (sent during UAbort_3)

R1 a: Compute and broadcast $\mathcal{D}_3$ (using p_{33}) on behalf of P_3 according to the protocol. Send $\{s_3, p_{31}, p_{33}, \mathrm{o}_3, \{\mathrm{o}_{3\alpha}^b\}_{\alpha\in[6\ell],b\in\{0,1\}}\}$ to P_1^*.

R1 a: Choose x_{21} at random and send to P_1^* on behalf of P_2.

R1 A: Sample pp_2 to compute $(\mathrm{c}_{21}, \mathrm{o}_{21}) \leftarrow \mathsf{Com}(\mathsf{pp}_2, x_{21})$. Broadcast $\{\mathsf{pp}_2, \mathrm{c}_{21}, \mathrm{c}_{23}\}$ where c_{23} is commitment of dummy value and send o_{21} to P_1^* on behalf of P_2.

R1 a: Receive $\{s_1, p_{11}, p_{13}, \mathrm{o}_1, \{\mathrm{o}_{1\alpha}^b\}_{\alpha\in[6\ell],b\in\{0,1\}}\}$ from P_1^* on behalf of P_3. Do all the verifications as an honest P_3 would perform for P_1 and update flag_2 with respect to (wrt) P_3.

R1 A: Set $\mathsf{flag}_2 = 1$ on behalf of P_3 if $p_{11} \neq x_{13}$ (x_{13} received in UAbort_1)

R1 a: Set $\mathsf{flag}_2 = 1$ on behalf of P_2 if P_1^* sends encoded inputs inconsistent with $\mathcal{D}_1$

R1 A: Set $\mathsf{flag}_2 = 1$ on behalf of P_2 if $\mathsf{Open}(\mathrm{c}_{12}, \mathrm{o}_{12}) \neq x_{12}$ or $m_{11} \neq x_{12}$ (x_{12} received during UAbort_1).

R2 a: On behalf of P_3: If $\mathsf{flag}_2 = 0$ wrt P_3, choose random z_3 and broadcast $\mathcal{W}_3$ as per the protocol. Else broadcast `abort`.

R2 a: On behalf of P_2: If $\mathsf{flag}_2 = 0$ wrt P_2, broadcast z_1, z_3 where z_1 is computed as per the protocol as $z_1 = x_{21} \oplus r_1$, where x_{21} sent to P_1^* in Round 1 and r_1 received from P_1^*. z_3 is either same as chosen above (if $\mathsf{flag}_2 = 0$ wrt P_3) or random (if $\mathsf{flag}_2 = 1$ wrt P_3). Else broadcast `abort`.

R2 a: Set $\mathsf{flag}_2 = 0$ on behalf of both P_2, P_3 if (a) `abort` was sent or received via broadcast in Round 2 (b) P_1^* broadcasts anything other than $(z_1, \mathrm{o}_{1(4\ell+\alpha)}^{z_1^\alpha})$ ($\mathrm{o}_{1(4\ell+\alpha)}^{z_1^\alpha}$ known on behalf of P_3) where $z_1 = x_{21} \oplus r_1$ (r_1, x_{21} known to P_2)

$\mathcal{S}_{\mathsf{UAbort}}$ **after** $\mathsf{UAbort}_1, \mathsf{UAbort}_2, \mathsf{UAbort}_3$

If $\mathsf{flag}_i = 1$ (on behalf of both P_2, P_3) for any $i \in [3]$, invoke $\mathcal{F}_{\mathsf{ua}}$ with `abort` on behalf of P_1^*. Else invoke $\mathcal{F}_{\mathsf{ua}}$ with `continue` on behalf of P_1^*.

Figure 3.11: Simulator $\mathcal{S}_{\mathsf{UAbort}}$

We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{ua}},\mathcal{S}_{\mathsf{UAbort}}} \overset{c}{\approx} \text{REAL}_{\mathsf{UAbort},\mathcal{A}}$, when $\mathcal{A}$ corrupts P_1. The views are shown to be indistinguishable via a series of intermediate hybrids.

- HYB_0: Same as $\text{REAL}_{\mathsf{UAbort},\mathcal{A}}$.
- HYB_1: Same as HYB_0, except that P_2, P_3 in UAbort_1 use uniform randomness rather than pseudo-randomness for the garbled circuit construction.

– HYB_2: Same as HYB_1, except that some of the commitments of encoded inputs which will not be sent to P_1 during UAbort_1 are replaced with commitment on dummy values. Specifically, these are corresponding to indices not equal to $m_{22}, m_{23}, r_2, r_3, z_2, z_3$ for C_2 and not equal to $m_{32}, m_{33}, r_2, r_3, z_2, z_3$ for C_3.

– HYB_3 : Same as HYB_2, except that when the execution results in P_1 evaluating GCs during UAbort_1, the GC C_2 is created as $(\mathsf{C}_2', d_2) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \mathsf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{r_2^\alpha}, e_{2(3\ell+\alpha)}^{r_3^\alpha}, e_{2(4\ell+\alpha)}^{z_2^\alpha}, e_{2(5\ell+\alpha)}^{z_3^\alpha}\}_{\alpha\in[\ell]})$. The commitment c_2 is later equivocated to C_2' using o_2 computed via $\mathrm{o}_2 \leftarrow \mathsf{Equiv}(\mathrm{c}_2, \mathsf{C}_2', t_2)$. The set of ciphertexts ct generated uses d_2 in their keys.

– HYB_4 : Same as HYB_3, except that when the execution results in P_1 evaluating GCs during UAbort_1, the GC C_3 is created as $(\mathsf{C}_3', d_3) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \mathsf{X}_3 = \{e_{3\alpha}^{m_{32}^\alpha}, e_{3(\ell+\alpha)}^{m_{33}^\alpha}, e_{3(2\ell+\alpha)}^{r_2^\alpha}, e_{3(3\ell+\alpha)}^{r_3^\alpha}, e_{3(4\ell+\alpha)}^{z_2^\alpha}, e_{3(5\ell+\alpha)}^{z_3^\alpha}\}_{\alpha\in[\ell]})$. The commitment c_3 is later equivocated to C_3' using o_3 computed via $\mathrm{o}_3 \leftarrow \mathsf{Equiv}(\mathrm{c}_3, \mathsf{C}_3', t_3)$. The set of ciphertexts ct generated uses d_3 in their keys.

– HYB_5: Same as HYB_4, except that during UAbort_2, flag_2 is set to 1 if $\mathcal{W}_1$ broadcast by P_1 has anything other than (opening of) encoded input corresponding to z_1 in C_1.

– HYB_6: Same as HYB_5, except that during UAbort_3, flag_3 is set to 1 if $\mathcal{W}_1$ broadcast by P_1 has anything other than (opening of) encoded input corresponding to z_1 in C_1.

– HYB_7: Same as HYB_6, except that when the execution does not result in P_1 getting access to the opening of commitment c_{23} (corresponding to x_{23}) broadcast by P_2 during UAbort_2, the commitment is replaced with commitment of dummy value.

– HYB_8: Same as HYB_7, except that when the execution does not result in P_1 getting access to the opening of commitment c_{32} (corresponding to x_{32}) broadcast by P_3 during UAbort_3, the commitment is replaced with commitment of dummy value.

– HYB_9: Same as HYB_8, except that when the execution UAbort_1 does not result in P_1 getting conflicting output on two garbled circuits, the set of ct is replaced by encryption of a dummy message.

Since $\text{HYB}_9 := \text{IDEAL}_{\mathcal{F}_{\mathsf{ua}}, \mathcal{S}_{\mathsf{UAbort}}}$, we show that every two consecutive hybrids are computationally indistinguishable which concludes the proof.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that P_2, P_3 in UAbort_1 use uniform randomness in HYB_1 rather than pseudorandomness as in HYB_0. The indistinguishability follows

via reduction to the security of the PRG G.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is some of the commitments of encoded inputs which will not be sent to P_1 during UAbort_1 are replaced with commitment on dummy messages. The indistinguishability follows from the hiding property of NICOM.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is in the way $(\mathsf{C}_2, \mathsf{X}, \mathsf{d}_2)$ is generated. In HYB_2, $(\mathsf{C}_2, \mathsf{e}_2, \mathsf{d}_2) \leftarrow \mathsf{Gb}(1^\kappa, C)$ is run, which gives $(\mathsf{C}_2, \mathsf{En}(x, \mathsf{e}), \mathsf{d}_2)$. In HYB_3, it is generated as $(\mathsf{C}_2', \mathsf{d}_2) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \mathsf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{r_2^\alpha}, e_{2(3\ell+\alpha)}^{r_3^\alpha}, e_{2(4\ell+\alpha)}^{z_2^\alpha}, e_{2(5\ell+\alpha)}^{z_3^\alpha}\}_{\alpha\in[\ell]})$. The commitment to the garbled circuit is later equivocated to C_2' using o_2 computed via $\mathrm{o}_2 \leftarrow \mathsf{Equiv}(\mathrm{c}_2, \mathsf{C}_2', t_2)$. The indistinguishability follows via reduction to the privacy of the garbling scheme and the hiding property of eCom.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: Similar argument as above with respect to C_3.

$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_5$: The difference between the hybrids is that in HYB_4, flag_2 is set to 1 if $\mathcal{W}_1$ broadcast by P_1 during UAbort_2 has (opening of) encoded input that is inconsistent with commitment corresponding to z_1 in $\mathcal{D}_1$, while in HYB_5, flag_2 is set to 1 if $\mathcal{W}_1$ broadcast by P_1 has (opening of) encoded input anything other than encoding of z_1 corresponding to C_1. It follows from the binding property of NICOM that P_1 could have sent an encoded input not consistent with C_1 but consistent with $\mathcal{D}_1$, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_5 \overset{c}{\approx} \text{HYB}_6$: Similar argument as above with respect to $\mathcal{W}_1$ broadcast by P_1 during UAbort_3.

$\text{HYB}_6 \overset{c}{\approx} \text{HYB}_7$: The difference between the hybrids is that when the execution does not result in P_1 getting access to the opening of commitment c_{23} (corresponding to x_{23}) broadcast by P_2 during UAbort_2, c_{23} corresponds to the actual input share x_{23} in HYB_8 while it corresponds to dummy value in HYB_9. The indistinguishability follows from the hiding property of NICOM Com.

$\text{HYB}_7 \overset{c}{\approx} \text{HYB}_8$: Similar argument as above with respect to commitment c_{32} broadcast by P_3 during UAbort_3.

$\text{HYB}_8 \overset{c}{\approx} \text{HYB}_9$: The difference between the hybrids is that when the execution UAbort_1 does

not result in P_1 getting conflicting output on two garbled circuits, in HYB_8, the set of ct is the encryption of opening of shares of input while in HYB_9, it is replaced with encryption of dummy message. Assuming the encryption key is unknown to P_1 (holds except with negligible probability due to authenticity), indistinguishability follows from the CPA security of the encryption scheme.

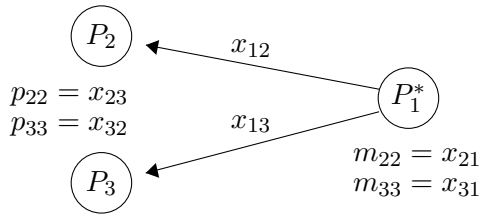
3.7.3 Proof of Security for Protocol GOD

In this section, we present the proof of security of GOD relative to the ideal functionality for god (Figure 2.4). For better clarity, we assume without loss of generality that P_1 is corrupt (denoted as P_1^*) and describe the simulator $\mathcal{S}_{\mathsf{GOD}}$. Since the roles of the parties are symmetric in GOD, similar proof would hold in case of corrupt P_2, P_3 as well. The simulator plays the role of the honest parties P_2, P_3 and simulates each step of the protocol GOD.

Similar to $\mathcal{S}_{\mathsf{UAbort}}$, we divide the description of $\mathcal{S}_{\mathsf{GOD}}$ as follows: We describe $\mathcal{S}_{\mathsf{GOD}}$ during GOD_1 where corrupt P_1^* is the evaluator and during GOD_2 when corrupt P_1^* acts as a garbler. The steps corresponding to GOD_3, would follow symmetrically from that described corresponding to GOD_2. We then describe the steps of the simulator $\mathcal{S}_{\mathsf{GOD}}$ corresponding to the third round. In the protocol GOD, the behavior of corrupt P_1 in Round 1, 2 determines his committed input. Hence, the privacy simulator can only be invoked earliest after the simulation of the first round. Similar to $\mathcal{S}_{\mathsf{UAbort}}$, since the commitments on encoding information is sent in the first round itself, we use a modified version of the privacy simulator of the garbling scheme which additionally takes an encoded input as parameter (see Section 3.7.2). The simulator $\mathcal{S}_{\mathsf{GOD}}$ appears in Figure 3.12 with **R1/R2/R3** indicating simulation for round 1, 2 and 3 and and **g/G** denoting the steps corresponding to subprotocol $\mathsf{GOD}_i, \mathsf{GOD}$ respectively.

Simulator $\mathcal{S}_{\mathsf{GOD}}$

$\mathcal{S}_{\mathsf{GOD}}$ **during** $\mathsf{GOD}_1, \mathsf{GOD}$



R1 g: Receive x_{12}, x_{13} from P_1^* on behalf of P_2, P_3.

R1 G: Receive $(\mathsf{pp}_1, \mathsf{c}_{12}, \mathsf{c}_{13})$ via broadcast and $(\mathsf{o}_{12}, \mathsf{o}_{13})$ privately from P_1^* on behalf of P_2, P_3. Set

$\mathcal{C}_i = \{P_1\}$ if $\mathsf{Open}(\mathrm{c}_{1i}, \mathrm{o}_{1i}) \neq x_{1i}$ for $i \in \{2,3\}$

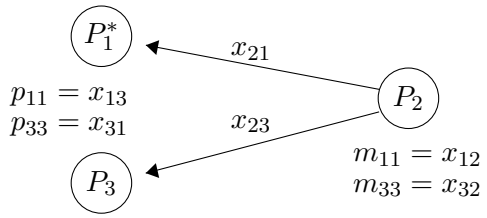
R1 g: Sample $\mathsf{epp}_2, \mathsf{epp}_3$ for eCom, having trapdoor t_2, t_3. Choose $m_{22} = x_{21}$ (sent during GOD_2), $m_{33} = x_{31}$ (sent during GOD_3), m_{23}, m_{32} at random. On behalf of P_i ($i \in \{2,3\}$) do the following: compute $(\mathsf{C}_i, e_i, d_i) \leftarrow \mathsf{Gb}(1^\kappa, C)$ using uniform randomness. Broadcast $\mathcal{D}_i = (\mathsf{epp}_i, \mathsf{pp}_i, \mathrm{c}_i, \{\mathrm{c}^b_{i\alpha}\}_{\alpha\in[4\ell],b\in\{0,1\}})$ where c_i, $\{\mathrm{c}^{m^\alpha_{i2}}_{i\alpha}, \mathrm{c}^{m^\alpha_{i3}}_{i(\ell+\alpha)}, \mathrm{c}^0_{i(2\ell+\alpha)}, \mathrm{c}^1_{i(2\ell+\alpha)}, \mathrm{c}^0_{i(3\ell+\alpha)}, \mathrm{c}^1_{i(3\ell+\alpha)}\}_{\alpha\in[\ell]}$ be computed as as per the protocol. Let the remaining $\{\mathrm{c}^b_{i\alpha}\}$ commit to dummy values. (For Naor-based eNICOM, c_i set to the specific commitment that suppports equivocation)

R2 g: If $P_1 \notin \mathcal{C}_2, \mathcal{C}_3$, invoke $\mathcal{F}_{\mathsf{god}}$ with $(\mathsf{sid}, \mathsf{Input}, x_1)$ on behalf of P_1^* to obtain output y, where $x_1 = x_{12} \oplus x_{13}$. For ($i \in \{2,3\}$), run $(\mathsf{C}'_i, d_i) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \mathsf{X}_i = \{e^{m^\alpha_{i2}}_{i\alpha}, e^{m^\alpha_{i3}}_{i(\ell+\alpha)}, e^{x^\alpha_{12}}_{i(2\ell+\alpha)}, e^{x^\alpha_{13}}_{i(3\ell+\alpha)}\}_{\alpha\in[\ell]})$. Using trapdoor t_i, compute $\mathrm{o}_i = \mathsf{Equiv}(\mathrm{c}_i, \mathsf{C}'_i, t_i)$. Send $\mathtt{OK}$ msg on behalf of P_2, P_3 as per the protocol using computed $\mathrm{o}_2, \mathrm{o}_3$.

R2 g: Else if $P_1 \notin \mathcal{C}_i$ for $i \in \{2,3\}$, act on behalf of P_i as per the protocol opening the garbled circuit (equivocate c_i to C_i in case of Naor-based eNICOM) and encoded input as per m_{23} and m_{32}

R2 g: If $P_1 \notin \mathcal{C}_i$ ($i \in \{2,3\}$), send set of ct on behalf of P_i using a dummy message.

$\mathcal{S}_{\mathsf{GOD}}$ **during** $\mathsf{GOD}_2, \mathsf{GOD}$



R1 G: Set $p_{33} = x_{31}$ (sent during GOD_3) on behalf of P_3.

R1 g: Compute and broadcast $\mathcal{D}_3$ (using p_{33}) on behalf of P_3 and send private information to P_1^* as per protocol

R1 G: Compute $(\mathrm{c}_{21}, \mathrm{o}_{21}) \leftarrow \mathsf{Com}(\mathsf{pp}_2, x_{21})$ with randomly chosen x_{21}. Broadcast $\{\mathsf{pp}_2, \mathrm{c}_{21}, \mathrm{c}_{23}\}$ where c_{23} is commitment of dummy value

R1 g: Send $\{x_{21}, \mathrm{o}_{21}\}$ to P_1^* on behalf of P_2.

R1 g: Do all the verifications wrt $\mathcal{D}_1$ as an honest P_3 would perform for P_1 and update $\mathcal{C}_3$.

R1 G: Add P_1 to $\mathcal{C}_2$ if $m_{11} \neq x_{12}$.

R2 g: Add P_1 to $\mathcal{C}_2$ if any of the openings sent by P_1 (for C_3 or encoded inputs) is anything other than originally committed (known on behalf of P_3).

R2 g: If $P_1 \notin \mathcal{C}_2$ and $P_1 \in \mathcal{C}_3$: Extract P_1's input x_1 if committed: **(a)** either on clear with `nOK` **(b)** or in encoded form as $x_1 = m_{31} \oplus p_{31}$. Invoke $\mathcal{F}_{\mathsf{god}}$ with $(\mathsf{sid}, \mathsf{Input}, x_1)$ on behalf of P_1^* to obtain output y. Else, (P_1's input not committed) invoke $\mathcal{F}_{\mathsf{god}}$ with $(\mathsf{sid}, \mathsf{Input}, x_1)$ on behalf of P_1^* to obtain output y for default x_1.

$\mathcal{S}_{\mathsf{GOD}}$ during R3

If $P_1 \in \mathcal{C}_2, \mathcal{C}_3$ at the end of round 1, invoke $\mathcal{F}_{\mathsf{god}}$ with $(\mathsf{sid}, \mathsf{Input}, x_1)$ on behalf of P_1^* to obtain y for a default x_1. Send y to P_1^* on behalf of both P_2 and P_3 if $P_1 \in \mathcal{C}_2, \mathcal{C}_3$ in the end of round one. Send y to P_1^* on behalf of only P_2 (P_3) if $P_1 \in \mathcal{C}_3$ ($\mathcal{C}_2$) in the end of round one.

Figure 3.12: Description of $\mathcal{S}_{\mathsf{GOD}}$

We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}_{\mathsf{GOD}}} \overset{c}{\approx} \text{REAL}_{\mathsf{GOD}, \mathcal{A}}$, when $\mathcal{A}$ corrupts P_1. The views are shown to be indistinguishable via a series of intermediate hybrids.

- HYB_0: Same as $\text{REAL}_{\mathsf{GOD}, \mathcal{A}}$.
- HYB_1: Same as HYB_0, except that P_2, P_3 in GOD_1 use uniform randomness rather than pseudo-randomness for the garbled circuit construction.
- HYB_2: Same as HYB_1, except that some of the commitments that will not be opened by P_1 during GOD_1 are replaced with commitment on dummy values. Specifically, these are corresponding to indices not equal to $m_{22}, m_{23}, x_{12}, x_{13}$ for C_2 and not equal to $m_{32}, m_{33}, x_{12}, x_{13}$ for C_3.
- HYB_3 : Same as HYB_2, except that when the execution results in P_1 evaluating GCs during GOD_1, the GC C_2 is created as $(\mathsf{C}_2', d_2) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \mathsf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{x_{12}^\alpha}, e_{2(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha \in [\ell]})$. The commitment c_2 is later equivocated to C_2' using o_2 computed via $\mathrm{o}_2 \leftarrow \mathsf{Equiv}(\mathrm{c}_2, \mathsf{C}_2', t_2)$. The set of ciphertexts ct generated uses d_2 in their keys.
- HYB_4 : Same as HYB_3, except that when the execution results in P_1 evaluating GCs during GOD_1, the GC C_3 is created as $(\mathsf{C}_3', d_3) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \mathsf{X}_3 = \{e_{3\alpha}^{m_{32}^\alpha}, e_{3(\ell+\alpha)}^{m_{33}^\alpha}, e_{3(2\ell+\alpha)}^{x_{12}^\alpha}, e_{3(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha \in [\ell]})$. The commitment c_3 is later equivocated to C_3' using o_3 computed via $\mathrm{o}_3 \leftarrow \mathsf{Equiv}(\mathrm{c}_3, \mathsf{C}_3', t_3)$. The set of ciphertexts ct generated uses d_3 in their keys.
- HYB_5: Same as HYB_4, except that during GOD_2, $\mathcal{C}_2$ is set to P_1 if P_2 receives o_3 that opens to a value other than the originally committed C_3.

– HYB_6: Same as HYB_5, except that during GOD_3, $\mathcal{C}_3$ is set to P_1 if P_3 receives o_2 that opens to a value other than the originally committed C_2.

– HYB_7: Same as HYB_6, except that during GOD_2, $\mathcal{C}_2$ is set to P_1 if P_2 accepts any encoded input not consistent with $\mathsf{C}_1, \mathsf{C}_3$

– HYB_8: Same as HYB_7, except that during GOD_3, $\mathcal{C}_3$ is set to P_1 if P_3 accepts any encoded input not consistent with $\mathsf{C}_1, \mathsf{C}_2$

– HYB_9: Same as HYB_8, except that when the execution does not result in P_1 getting access to the opening of commitment c_{23} (corresponding to x_{23}) broadcast by P_2 during GOD_2, the commitment is replaced with commitment of dummy value.

– HYB_{10}: Same as HYB_9, except that when the execution does not result in P_1 getting access to the opening of commitment c_{32} (corresponding to x_{32}) broadcast by P_3 during GOD_3, the commitment is replaced with commitment of dummy value.

– HYB_{11}: Same as HYB_{10}, except that when the execution GOD_1 does not result in P_1 getting conflicting output on two garbled circuits, the set of ct is replaced by encryption of a dummy message.

Since $\text{HYB}_{11} := \text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}_{\mathsf{GOD}}}$, we show that every two consecutive hybrids are computationally indistinguishable which concludes the proof.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that P_2, P_3 in GOD_1 use uniform randomness in HYB_1 rather than pseudorandomness as in HYB_0. The indistinguishability follows via reduction to the security of the PRG G.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is some of the commitments that will not be opened by P_1 during GOD_1 are replaced with commitments on dummy values. The indistinguishability follows from the hiding property of the commitment scheme.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is in the way $(\mathsf{C}_2, \mathsf{X}, \mathsf{d}_2)$ is generated. In HYB_2, $(\mathsf{C}_2, \mathsf{e}_2, \mathsf{d}_2) \leftarrow \mathsf{Gb}(1^\kappa, C)$ is run, which gives $(\mathsf{C}_2, \mathsf{En}(x, \mathsf{e}), \mathsf{d}_2)$. In HYB_3, it is generated as $(\mathsf{C}_2', \mathsf{d}_2) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y, \mathsf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{x_{12}^\alpha}, e_{2(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha\in[\ell]})$. The commitment to the garbled circuit is later equivocated to C_2' using o_2 computed via $\mathrm{o}_2 \leftarrow \mathsf{Equiv}(\mathrm{c}_2, \mathsf{C}_2', t_2)$. The indistinguishability follows via reduction to the privacy of the garbling scheme and the hiding

property of eCom.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: Similar argument as above with respect to C_3.

$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_5$: The difference between the hybrids is that in HYB_4, P_2 sets $\mathcal{C}_2 = P_1$ if the o_3 sent by P_1 in GOD_2 output $\perp$ while in HYB_5, P_2 sets $\mathcal{C}_2 = P_1$ if o_3 sent by P_1 in GOD_2 opens to any value other than C_3. Since the commitment scheme eCom is binding and epp was chosen uniformly at random by P_3, in HYB_4, P_1 could have decommitted successfully to a different garbled circuit than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_5 \overset{c}{\approx} \text{HYB}_6$: Similar argument as above with respect to P_3 in GOD_3.

$\text{HYB}_6 \overset{c}{\approx} \text{HYB}_7$: The difference between the hybrids is that in HYB_6, P_2 sets $\mathcal{C}_2 = P_1$ if opening of commitment on the encoded inputs sent by P_1 in GOD_2 results in $\perp$ while in HYB_7, $\mathcal{C}_2$ is set to P_1 if P_2 accepts the opening of any commitment to a value other than what was originally committed. The indistinguishability between the hybrids follows from the binding property of NICOM.

$\text{HYB}_7 \overset{c}{\approx} \text{HYB}_8$: Similar argument as above with respect to P_3 in GOD_3.

$\text{HYB}_8 \overset{c}{\approx} \text{HYB}_9$: The difference between the hybrids is that when the execution does not result in P_1 getting access to the opening of commitment c_{23} (corresponding to x_{23}) broadcast by P_2 during GOD_2, c_{23} corresponds to the actual input share x_{23} in HYB_8 while it corresponds to dummy value in HYB_9. The indistinguishability follows from the hiding property of Com.

$\text{HYB}_9 \overset{c}{\approx} \text{HYB}_{10}$: Similar argument as above with respect to commitment c_{32} broadcast by P_3 during GOD_3.

$\text{HYB}_{10} \overset{c}{\approx} \text{HYB}_{11}$: The difference between the hybrids is that when the execution GOD_1 does not result in P_1 getting conflicting output on two garbled circuits, in HYB_{10}, the set of ct is the encryption of an input and a share of input while in HYB_{11}, it is replaced with encryption of dummy message. Assuming the encryption key is unknown to P_1 (holds except with negligible probability due to authenticity), indistinguishability follows from the CPA security of the encryption scheme.

3.8 Appendix: Authenticated Conditional Disclosure of Secret

The subprotocol Cert_i (Figure 3.2) used in our protocol Fair is reminiscent of the notion of 'conditional disclosure of secrets (CDS)' which was first introduced in [103]. Informally, the problem of conditional disclosure of secrets involves two parties Alice and Bob, who hold inputs x and y respectively and wish to release a common secret s to Carol (who knows both x and y) if only if the input (x, y) satisfies some predefined predicate f. The model allows Alice and Bob to have access to shared random string (hidden from Carol) and the only communication allowed is a single unidirectional message sent from each player (Alice and Bob) to Carol. Traditionally, CDS involves two properties, namely *correctness* (if $f(x, y)$ is true, then Carol is always able to reconstruct s from her input and the messages she receives) and *privacy* (if $f(x, y)$ is false, Carol obtains no information about the secret s). Formally,

Definition 3.2 (Conditional Disclosure of Secret) *[5] Let $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ be a predicate. Let $F_1 : \mathcal{X} \times \mathcal{S} \times \mathcal{R} \to \mathcal{T}_1$ and $F_2 : \mathcal{Y} \times \mathcal{S} \times \mathcal{R} \to \mathcal{T}_2$ be deterministic encoding algorithms, where $\mathcal{S}$ is the secret domain. Then, the pair (F_1, F_2) is a CDS scheme for f if the function $F(x, y, s, r) = (F_1(x, s, r), F_2(y, s, r))$ that corresponds to the joint computation of F_1 and F_2 on a common s and r, satisfies the following:*

- *δ-correctness: There exists a deterministic algorithm Dec, called a decoder, such that for every 1-input (x, y) of f and any secret $s \in \mathcal{S}$, the following holds: $Pr_{r \leftarrow \mathcal{R}}[\mathsf{Dec}(x, y, F(x, y, s, r)) \neq s] \leq \delta$*
- *ϵ-privacy: There exists a simulator $\mathcal{S}$ such that for every 0-input (x, y) of f and any secret $s \in \mathcal{S}$, it holds that $|Pr[D(\mathcal{S}(x, y) = 1)] - Pr[D(F(x, y, s, r)) = 1]| \leq \epsilon$ for every distinguisher D. ($\mathcal{S}, D$ assumed to be poly-time or computationally unbounded depending on computational / information-theoretic setting).*

Interestingly, we find that the functionality realized by subprotocol Cert_i subsumes the above properties under computational variant adapted to tolerate active corruption of single party and gives some stronger guarantees. We thus formally define a variant of CDS known as 'Authenticated Conditional Disclosure of Secret' below and show realization of the same by Cert_i.

Definition 3.3 (Authenticated Conditional Disclosure of Secret) *Let A, B denote two parties holding inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ respectively and having access to common secret $s \in \mathcal{S}$*

and C denote an external party. We assume a PPT adversary $\mathcal{A}$ who can actively corrupt at most 1 party among A, B and C. An authenticated CDS protocol is secure against $\mathcal{A}$ if the following properties hold:

- *δ-correctness holds for honest A, B, and C where $\delta = \mathtt{negl}(\kappa)$.*
- *ϵ-privacy holds against $\mathcal{A}$ corrupt C, where $\epsilon = \mathtt{negl}(\kappa)$.*
- *Authenticity: For 1-input (x, y) of f and any secret s, Dec may result in $\perp$ when either A or B is corrupt, in which case C either identifies a corrupt party or a pair of parties in conflict that includes the corrupt party.*

Our Cert_i gives an authenticated CDS as follows. The garblers P_j, P_k take the role of A and B and the evaluator takes the role of C. The common randomness r is the seed for the PRG used for generating the entire randomness for GC generation etc. The secret s is the key corresponding to 1 in the circuit. The predicate is the circuit that we garble in Cert_i. While for the purpose of our 3-round fair protocol, the predicate is equality checking, in theory, we can garble any predicate. F_1 and F_2 are the codes of P_j and P_k respectively. Dec is the code that P_i executes. The correctness and privacy follow from the correctness and authenticity of the garbling scheme. The authenticity follows from the fact that P_i either receives the correct secret or detects a conflict or corrupt.

Chapter 4

Fast Secure Computation for 3PC and 4PC over the Internet

In this chapter, we present efficient, *constant-round* 3-party (3PC) and 4-party (4PC) protocols in the honest-majority setting that achieve strong security notions of fn and god. Being constant-round, our constructions are suitable for Internet-like high-latency networks and are built from garbled circuits (GC).

4.1 Introduction

While the earlier works in MPC literature traditionally been focused on theoretical aspects, lately, with increasing demand for efficient constructions suitable in real-time applications, there has been a growing interest to improve the concrete efficiency of protocols. The domain of MPC can be broadly classified into honest majority [30, 177, 19, 18, 72, 33, 20, 21] and dishonest majority [107, 73, 34, 76, 11, 94, 151] settings. The special case of two-party (2PC) in dishonest majority setting has enjoyed overwhelming focus over the years in terms of improving its efficiency [147, 150, 145, 2, 163]. In contrast, the special cases of honest majority setting have not been in the limelight until recently when practically efficient MPC constructions of [52, 159, 7, 91] leveraged presence of small number of parties. Having honest majority is not only advantageous since it enables strong desirable security notions of fn and god but also since it allows us to obtain constructions relying on weaker cryptographic assumptions and light-weight cryptographic tools. For example, the protocols of [129, 159] are built using symmetric-key primitives whereas 2PC protocols require Oblivious Transfer (OT) [182, 147, 125]. In this work, we consider the honest-majority setting for small number of parties ($n = 3$ and $n = 4$) tolerating at most one malicious corruption ($t = 1$). While most works outlined in Section 1.3

considered round-optimal protocols, we outline the relevant literature related to constant-round efficient MPC with small number of parties beyond the two-party case below.

4.1.1 Related Work.

The regime of MPC over small population has seen growth both in the domain of low-latency and high-throughput protocols. Relying on garbled circuits, the unique selling point of the former is constant rounds and these serve better in high-latency networks such as the Internet. Whereas, the added edge of the latter category is low communication overhead (band-width) and simple computations. Building on secret sharing, this category however takes number of rounds proportional to the depth of the circuit representing the function to be computed. These primarily cater to low-latency networks.

In the domain of constant-round protocols which is the focus of this work, [159] presents a 3-round efficient 3-party (3PC) protocol tolerating at most one malicious corruption and involving transmission and evaluation of a single garbled circuit. Concurrently, in the 3-party setting, [129] achieves a 2-round protocol whose cost is essentially that of 3 garbled circuits. However, both these protocols achieve a weaker notion of security i.e sa. In the presence of a broadcast channel, the 3PC of [159] can additionally achieve ua, albeit for specific class of functions that give same output to all. The work of [129] presents a 2-round 4-party (4PC) protocol tolerating single corruption that achieves god in the absence of broadcast channel. Since the focus of [129] is on minimizing the number of rounds of interaction, the protocol comprises of several parallel instances of private simultaneous message (PSMs) which when instantiated with garbled circuit (GC) would sum upto communication of 12 GCs. The recent work of [166] explores the exact round complexity of 3PC protocols under various security notions including fn and god. While the protocols are round-optimal, they involve a minimum of 3 GCs. The work of [52] explores the case of 5-party with two malicious corruptions and relies on distributed garbling approach of [29] (which is more expensive than Yao's garbling). Recent paper of [28], improving on the distributed garbling techniques of [29], proposes an honest majority protocol with $n > 3t$ and shows practical implementation for 31 parties. The results mentioned above are designed in the honest majority. [58] studies 3PC in dishonest majority setting. In summary, the most relevant work that is close to our work efficiency-wise is that of [159] which we compare with.

There have been a flurry of works in the high-throughput domain recently [7, 91, 6, 8]. In 3-party setting, [7] and [91, 109] presents semi-honest and maliciously secure protocols respectively that are extremely fast on standard hardware. [8] significantly improves over the protocol of [91], achieving the computation rate of 1.15 billion AND gates/second. In the 4 party setting, the work of [109] provides a construction that is secure against one malicious corruption based

on the dual execution approach. They incur communication of 1.5 bits per party per gate for boolean circuits and thus offer a performance that is 4.5 times better than that of [8]. [109] also includes protocol variants for achieving fn and god.

4.1.2 Our Results.

While our contributions appear in detail in Section 1.4.1.2, we give a quick summary below (illustrated in Tables 4.1, 4.2 as well).

Assuming the minimal model of pairwise-private channels, we present two protocols that involve computation and communication of a *single* GC– (a) a 4-round 3PC with fn, (b) a 5-round 4PC with god. Empirically, our protocols are on par with the best known 3PC protocol of [159] that only achieves sa, in terms of the computation time, LAN runtime, WAN runtime and communication cost. In fact, our 4PC outperforms the 3PC of [159] significantly in terms of per-party computation and communication cost. With an extra GC, we improve the round complexity of our 4PC to four rounds. The only 4PC in our setting, given by [129], involves 12 GCs. Assuming an additional broadcast channel, we present a 5-round 3PC with god that involves computation and communication of a *single* GC. A broadcast channel is inevitable in this setting for achieving god, owing to the impossibility result of [67]. The overall broadcast communication of our protocol is nominal and most importantly, is independent of the circuit size. This protocol too induces a nominal overhead compared to the protocol of [159].

Table 4.1: Theoretical and Empirical Comparison

Ref.	# Parties	# GCs	Rounds	Security	Broadcast
[159]	3	1	3	sa	✗
Our Work [46]	3	1	4	fn	✗
Our Work [46]	3	1	5	god	✓ [67]
[129]	4	12	2	god	✗
Our Work [46]	4	2	4	god	✗
Our Work [46]	4	1	5	god	✗

Table 4.2: Experimental Results

Ref.	Computation (ms)	LAN (ms)	WAN (s)	Communication (KB)
3PC with fn	0.06 – 0.16	0.03 – 0.8	0.21 – 0.5	5.63 – 10.74
4PC with god	0.19 – 2.61 (**g**)	0.17 – 2.45 (**g**)	0.02 (+.49) – 0.31 (+.52)	18.63 (−.01) – 500.56 (−.1) (**g**)
3PC with god	0.16 – 0.3	1.52 – 3	-	0.19 (+.02) – 0.46 (+.11)

Table 4.2 shows the overhead or gain (indicated by **g**) of our protocols compared to the 3PC of [159] in terms of *average* computation time, LAN runtime, WAN runtime and communication cost, where the average is taken over the number of parties and the range is taken over the

choice of circuits. The increase in the overhead or decrease in the gain for the worst case 5-round run of our 3PC and 4PC with god is shown in the bracket. With respect to our 4-round 4PC with god, in the worst case run, we save one round at the expense of one garbled circuit over our 5-round 4PC which amounts to a value in the range 72 KB – 1530 KB for the benchmark circuits.

Roadmap: The preliminaries appear in Section 4.2. Our efficient 3PC protocols achieving fn and god are presented in Section 4.3 and 4.6 respectively. The 4PC protocol with rounds 5 and 4 appear in Section 4.4 and 4.5 respectively. The experimental results are presented in Section 4.7. The security proofs of the four protocols appear in Sections 4.8.1 - 4.8.4.

4.2 Preliminaries

4.2.1 Model and Notations

We consider a set $\mathcal{P}$ of at most four PPT parties, denoted by P_1, P_2, P_3, P_4. We assume that any two parties are connected by pair-wise secure and authentic channels. We assume the existence of a broadcast channel only for the 3PC protocol achieving god. Our model assumes a PPT adversary $\mathcal{A}$, who can statically and maliciously corrupt at most one party out of the 3 or 4 parties. For any subset $\mathcal{X}$ of $\mathcal{P}$, $\mathsf{ind}(\mathcal{X})$ refers to the indexes of the parties. For example, when $\mathcal{X} = \{P_1, P_2\}$, then $\mathsf{ind}(\mathcal{X}) = \{1, 2\}$.

4.2.2 Primitives

In addition to the primitives defined in Chapter 2, we use the following:

Replicated Secret Sharing (RSS) [70, 131] We use a 3-party replicated secret sharing scheme private against one corruption (1-private). Informally, for a secret s to be shared over a boolean field $\mathbb{F}_2$, we randomly choose r_1, r_2 and compute r_3 such that $s = r_1 \oplus r_2 \oplus r_3$ (where $r_3 = s \oplus r_1 \oplus r_2$). We refer to r_1, r_2, r_3 as the three shares of s. Each of the 3 participating parties say P_1, P_2, P_3 are given access to two among the three shares i.e (r_2, r_3), (r_1, r_3) and (r_1, r_2) respectively. Reconstruction of s is possible by combining the shares held by any two among the three parties. However, given only the shares of a single party, the distribution of shares appears random and hence s remains private. We say that two parties say P_1, P_3 hold consistent shares if $r_2' = r_2$ where (r_2', r_3) are the shares held by P_1 and (r_1, r_2) are the shares held by P_3 [129].

Pre-Image Resistance Hash [178] Consider a hash function family H: $\mathcal{K} \times \mathcal{M} \rightarrow \mathcal{Y}$. The hash function H is said to be pre-image resistant if for all probabilistic polynomial-time adversaries $\mathcal{A}$, given $y = \mathsf{H}_k(x)$ where $k \in_R \mathcal{K}; x \in_R \{0,1\}^m$, $\Pr[x' \leftarrow \mathcal{A}(k, y) : \mathsf{H}_k(x') = y]$ is

negligible in κ, where $m = \mathsf{poly}(\kappa)$.

Collision-Resistant Hash [178] Consider a hash function family H': $\mathcal{K} \times \mathcal{M} \to \mathcal{Y}$. The hash function H' is said to be collision resistant if for all probabilistic polynomial-time adversaries $\mathcal{A}$, given H'_k where $k \in_R \mathcal{K}; x \in_R \{0,1\}^m$, $\Pr[(x, x') \leftarrow \mathcal{A}(k) : (x \neq x') \wedge \mathsf{H}'_k(x) = \mathsf{H}'_k(x')]$ is *negligible* in κ, where $m = \mathsf{poly}(\kappa)$.

4.3 3PC with fn

In this section, we present an efficient fair 3PC protocol that consumes 4 rounds in a network constituting of only pairwise-private channels. The starting point of our protocol is that of [159]. In the protocol of [159], P_1, P_2 act as garblers while P_3 acts as an evaluator. The garblers use common randomness to construct the same GC individually. Since at most one party can be corrupt, a comparison of GCs received from the garblers allows the evaluator P_3 to conclude its correctness. Besides, P_3 additively shares his input among the at the beginning of the protocol. This eliminates the need of oblivious transfer (OT) to transfer the evaluator's encoded input, as the garblers can directly send the encoded inputs corresponding to their own input as well as the share of P_3's input held by them. To force the garblers to input encoded inputs (the keys) that are consistent with the GCs, the following technique is adopted. Together with the GC, each garbler also generates the commitment to the encoding information using the common shared randomness and communicates to the evaluator. Again a simple check on whether the set of commitments are same for both the garblers allows to conclude their correctness. Now it is infeasible for the garblers to decommit the encoded input corresponding to their own input and the evaluator's share to something that are inconsistent to the GC without being caught. Following a common trick to hide the inputs of the garblers, the commitments on the encoding information corresponding to every bit of the garblers' input are sent in permuted order that is private to the garblers. Now if evaluation of the GC by P_3 is successful, P_3 computes the output using soft decoding on the encoded output Y. P_3 then sends Y to the garblers, enabling them to decode the output. For a function where all parties receive same output, depending upon whether Y is broadcast or sent over pairwise channel, the protocol achieves ua or sa respectively. Specifically, in the latter case when Y is sent over point-to-point channel, a corrupt P_3 may choose to send Y to only one of the garblers, thereby achieving sa.

In the protocol of [159], the only scenario in which fn is violated is when a malicious P_3 computes the output via soft decoding but chooses not to send (or sends wrong) encoded output Y to the garblers. At a high-level, we overcome this limitation by using *oblivious* garbling instead and withholding the decoding information d from P_3 until he forwards Y. Obliviousness ensures

that P_3 gets no information regarding output as long as d is unknown to him. A corrupt P_3 is forced to send Y to the garblers if he wants to learn the output, in which case at least one the garblers P_1, P_2 also learn the output. Authenticity ensures that P_3 cannot forge an encoded output $\mathsf{Y}' \neq \mathsf{Y}$ such that its decoding is valid. Even if P_3 chooses to abort, fn is achieved as no party learns the output. However, this new step gives rise to the following issues: (a) A corrupt garbler may send incorrect decoding information to an honest P_3 who forwarded Y; (b) A corrupt P_3 may send the correct encoded output Y (obtained by GC evaluation) to only one of the garblers. To tackle (a), the garblers are made to commit to the decoding information which P_3 can verify by means of cross-checking across garblers. The binding property of the commitment scheme prevents the corrupt garbler from lying about the decoding information later. The second issue is trivial to resolve with a broadcast channel. Without a broadcast channel, each garbler is made to forward the encoded output received from the evaluator to its co-garbler with a "proof" that he indeed received the encoded output from P_3. Without a proof, a corrupt garbler may "pretend" to have received the encoded output from honest P_3, whereas in reality P_3 was unable to evaluate the GC.

We facilitate this "proof" using a preimage-resistant cryptographic hash H function (alternately, one-way function can be used). In Round 1, each garbler P_i chooses a random value r_i (which will serve as the proof) and sends its digest $h_i = \mathsf{H}(r_i)$ to the other two parties, while it sends r_i only to P_3. In Round 2, each garbler P_i forwards the digest received from its co-garbler (in Round 1) to P_3. For each digest h_i, P_3 verifies its validity (whether $h_i = \mathsf{H}(r_i)$) and consistency (whether both garblers are in agreement with respect to h_i) and aborts in case the checks fail. If no abort has occurred, an honest P_3 who is able to obtain Y upon successful GC evaluation additionally sends the preimage of a garbler's digest with the fellow garbler. This preimage helps a garbler to convince its fellow garbler about the fact that Y (which is also valid) was received from P_3. When an honest P_3 was unable to evaluate GC, the property of pre-image resistance of the hash ensures that the corrupt garbler P_1 will not have access to *any* r_2' such that $\mathsf{H}(r_2') = h_2$ except with negligible probability. Therefore, he will not be able to fool his honest co-garbler P_2 to accept. On the flip side, consider a corrupt P_3 who sends Y to P_1 alone. If P_3 sends any proof, say r_2' to P_1 that verifies (may not be the same r_2 received from P_2; note that given r_2, it may be possible for corrupt P_3 to compute r_2' such that $\mathsf{H}(r_2') = h_2$ since we do not assume H is second-preimage resistant), then P_1 would check $\mathsf{H}(r_2') = h_2$ holds, accept the output, forward the proof and the output to P_2. Importantly, pre-image resistance suffices for an honest P_2 who hasn't received Y from P_3, to conclude that P_3 is corrupt upon receiving *any* r_2' (may not be equal to r_2 picked by him) from P_1 such that $\mathsf{H}(r_2') = h_2$. Thus, P_2 can simply accept output from P_1.

The protocol f3PC appears in Figure 4.1. We use an eNICOM (Section 2.4.2.1) to commit to the decoding information. This is due to a technicality that arises in the security proof explained in Section 4.8.1. Our proofs and proposed optimizations for f3PC which are incorporated in our implementation are explained subsequently. Lastly, the protocol f3PC cannot be naively extended to obtain god even in the presence of a broadcast channel (which is necessary due to [67]). When the evaluator fails to obtain the encoded output, there should be a way to compute the output which either seems to need more parties to enact the role of the evaluator and consequently involvement of more than one GCs or seems to require more than four rounds. We take the latter way-out and design a 5-round protocol in Section 4.6.

Protocol f3PC

Inputs: Party P_α has x_α for $\alpha \in [3]$.

Common Inputs: The circuit $C(x_1, x_2, x_3, x_4)$ that computes $f(x_1, x_2, x_3 \oplus x_4)$ where x_1, x_2, x_3, x_4 as well as function output belong to $\{0,1\}^\ell$ for $\ell \in \mathsf{poly}(\kappa)$. P_3 is assumed to be the evaluator and (P_1, P_2) as the garblers.

Output: $y = C(x_1, x_2, x_3, x_4) = f(x_1, x_2, x_3 \oplus x_4)$ or $\perp$.

Primitives: $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ that is correct, private, oblivious and authentic, a NICOM $(\mathsf{Com}, \mathsf{Open})$, an eNICOM $(\mathsf{eGen}, \mathsf{eCom}, \mathsf{eOpen}, \mathsf{Equiv})$, a PRG G and a preimage-resistant Hash H.

Round 1:

– P_1 chooses random seed $s \in_R \{0,1\}^\kappa$ for G and sends s to P_2.

– P_1 does the following (Similar steps will be executed by P_2): Sample t_1 corresponding to its share epp_1 for eNICOM. Compute $h_1 = \mathsf{H}(r_1)$, where r_1 is chosen uniformly at random. Send $\{\mathsf{epp}_1, h_1\}$ to P_2 and $\{h_1, r_1\}$ to P_3.

– P_3 samples pp for the NICOM and sends (x_{31}, pp) to P_1, (x_{32}, pp) to P_2.

Round 2:

– $P_i (i \in [2])$ does the following:

- Compute epp using epp_i and the share epp_j received from P_j $(j \in [2] \setminus i)$. Forward h_j received from P_j to P_3.
- Compute GC $(\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C)$ using randomness from $G(s)$. Assume $\{\mathsf{K}^0_\alpha, \mathsf{K}^1_\alpha\}_{\alpha \in [\ell]}$, $\{\mathsf{K}^0_{\ell+\alpha}, \mathsf{K}^1_{\ell+\alpha}\}_{\alpha \in [\ell]}$, $\{\mathsf{K}^0_{2\ell+\alpha}, \mathsf{K}^1_{2\ell+\alpha}\}_{\alpha \in [2\ell]}$ correspond to the encoding information for the input of P_1, P_2 and shares of P_3 respectively.

- Choose permutation strings $p_1, p_2 \in_R \{0,1\}^\ell$ for the garblers' input wires and generate commitments to e and d using randomness from $G(s)$. For $b \in \{0,1\}$, $(c^b_\alpha, \mathrm{o}^b_\alpha) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathsf{e}^{p^\alpha_1 \oplus b}_\alpha)$, $(c^b_{\ell+\alpha}, \mathrm{o}^b_{\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathsf{e}^{p^\alpha_2 \oplus b}_{\ell+\alpha})$ for $\alpha \in [\ell]$ and $(c^b_{2\ell+\alpha}, \mathrm{o}^b_{2\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathsf{e}^b_{2\ell+\alpha})$ for $\alpha \in [2\ell]$. Let $(\mathsf{c}, \mathsf{o}) \leftarrow \mathsf{eCom}(\mathsf{epp}, \mathsf{d})$. Set $\mathcal{B} = \big\{\mathsf{epp}, \mathsf{C}, \{c^b_\alpha\}_{\alpha\in[4\ell], b\in\{0,1\}}, c\big\}$.

– P_1 computes $m_1 = x_1 \oplus p_1$ and sends to P_3: $\mathcal{B}$, the openings of the commitments corresponding to (x_1, x_{31}) i.e $\{\mathrm{o}^{m^\alpha_1}_\alpha, \mathrm{o}^{x^\alpha_{31}}_{2\ell+\alpha}\}_{\alpha\in[\ell]}$ and m_1. Similarly, P_2 computes $m_2 = x_2 \oplus p_2$ and sends to P_3: $\mathcal{B}$, the openings of the commitments corresponding to (x_2, x_{32}) i.e $\{\mathrm{o}^{m^\alpha_2}_{\ell+\alpha}, \mathrm{o}^{x^\alpha_{32}}_{3\ell+\alpha}\}_{\alpha\in[\ell]}$ and m_2.

– P_3 does the following computation locally.

  - Abort if $\mathcal{B}$ or (h_1, h_2) received from P_1, P_2 is not identical or $\mathsf{H}(r_i) \neq h_i$ for some $i \in [2]$.
  - Abort if $(\mathsf{X}_1, \mathsf{X}_2, \mathsf{X}_{31}, \mathsf{X}_{32})$ contains $\perp$ where for $\alpha \in [\ell]$: $\mathsf{X}^\alpha_1 = \mathsf{Open}(\mathsf{pp}, c^{m^\alpha_1}_\alpha, \mathrm{o}^{m^\alpha_1}_\alpha)$, $\mathsf{X}^\alpha_2 = \mathsf{Open}(\mathsf{pp}, c^{m^\alpha_2}_{\ell+\alpha}, \mathrm{o}^{m^\alpha_2}_{\ell+\alpha})$, $\mathsf{X}^\alpha_{31} = \mathsf{Open}(\mathsf{pp}, c^{x^\alpha_{31}}_{2\ell+\alpha}, \mathrm{o}^{x^\alpha_{31}}_{2\ell+\alpha})$, $\mathsf{X}^\alpha_{32} = \mathsf{Open}(\mathsf{pp}, c^{x^\alpha_{32}}_{3\ell+\alpha}, \mathrm{o}^{x^\alpha_{32}}_{3\ell+\alpha})$.
  - Else set $\mathsf{X} = \mathsf{X}_1|\mathsf{X}_2|\mathsf{X}_{31}|\mathsf{X}_{32}$ and run $\mathsf{Y} \leftarrow \mathsf{Ev}(\mathsf{C}, \mathsf{X})$ for $\mathsf{C} \in \mathcal{B}$.

Round 3: If $\mathsf{Y} \neq \perp$, P_3 sends (Y, r_2) to P_1 and (Y, r_1) to P_2.

Round 4: P_i $(i \in [2])$ does the following: Let $(j \in [2]\setminus i)$. Execute $y \leftarrow \mathsf{De}(\mathsf{Y}, \mathsf{d})$, compute $z = \mathsf{H}(r'_j)$ if (Y, r'_j) received from P_3. If $y \neq \perp$ and $z = h_j$ (received from P_j in Round 1), send o to P_3 and (y, r'_j) to P_j. Else set $y = \perp$.
The parties do the following.

– P_3 runs $\mathsf{d} \leftarrow \mathsf{eOpen}(\mathsf{epp}, \mathsf{c}, \mathsf{o})$ where P_3 received o from P_i $(i \in [2])$. For $\mathsf{d} \neq \perp$, P_3 outputs $y \leftarrow \mathsf{De}(\mathsf{Y}, \mathsf{d})$.

– P_i $(i \in [2])$ does the following if $y = \perp$: If received (y', r'_i) from $P_{[2]\setminus i}$ such that $\mathsf{H}(r'_i) = h_i$, set $y = y'$.

Figure 4.1: Protocol f3PC

4.3.1 Correctness and Security

Theorem 4.1 *The protocol* f3PC *is correct.*

Proof: The inputs committed by P_3 is defined by the shares it distributes to the garblers in the first round. The inputs committed by the garblers are defined based on their openings of commitments. The encoded output obtained upon evaluation is based on the committed inputs. The correctness of the output follows from the correctness of the garbling scheme. □

While the formal proof is deferred to Section 4.8.1, we give intuition for fn and state the theorem below. We need to argue that a corrupt party gets the output of the computation if and only if the honest parties receive the output. For the forward direction assume that a corrupt party gets the output. Say the evaluator P_3 is corrupt. Due to oblivious garbling, P_3 would obtain the output only if given access to decoding information. This would occur only if he had sent a valid (Y, r_j) to at least one of the garblers say P_i (P_j is the co-garbler) i.e., $\mathsf{De}(\mathsf{Y}, \mathsf{d}) \neq \perp$ and $\mathsf{H}(r_j) = h_j$. P_i would communicate (Y, r_j) to P_j as well which would be verified and subsequently accepted by P_j. Thus all parties would learn the output. The case of corrupt garbler, say P_1 obtaining the output is straightforward - it would occur only in the case when the honest P_3 is able to evaluate the garbled circuit successfully. In this case, it is easy to see that the honest garbler P_2 and evaluator P_3 would be able to obtain the output using encoded output and decoding information received from the other respectively.

For the opposite direction, suppose an honest P_3 gets the output. Both garblers must have obtained the output via the encoded output sent by P_3. Finally an honest garbler, say P_1 who gets the output by decoding Y received from P_3, would forward the decoding information enabling P_3 to get the output as well. Next, an honest P_1 would accept the output only if he has a valid proof r_2' corresponding to his co-garbler P_2 i.e $\mathsf{H}(r_2') = h_2$. This proof would be verified and output accepted by P_2. This completes the intuition.

Theorem 4.2 *If one way functions exists, then protocol* $\mathsf{f3PC}$ *securely realizes the functionality* $\mathcal{F}_{\mathsf{fair}}$ *(Figure 2.3) against a malicious adversary that corrupts at most one party.*

4.3.2 Optimizations and generalization

We propose the following optimizations to improve communication efficiency. Firstly, P_1 and P_2 treat the common message $\mathcal{B}$ sent privately to P_3 in Round 2 as a string $\mathcal{B}$, divided into equal halves $\mathcal{B} = \mathcal{B}^1||\mathcal{B}^2$. P_1 sends $\mathcal{B}^1$ and $\mathsf{H}'(\mathcal{B}^2)$ while P_2 sends $\mathsf{H}'(\mathcal{B}^1)$ and $\mathcal{B}^2$ to P_3, where H' refers to a collision-resistant hash function (definition in Section 4.2) This would suffice for P_3 to verify if P_1, P_2 agree on a common $\mathcal{B}$. This optimization technique not only reduces the communication, but also improves the latency (transmission time) when both P_1, P_2 run at the same time [159]. The second optimization is to use equivocal commitment on the hash of the decoding information (collision-resistant hash), rather than simply committing on the decoding information.

Our protocol design has a natural extension to more than 3 parties (still for one corruption) without inflating the round complexity and number of GCs. The generalized protocol comprises of $(n-1)$ garblers who use common randomness for garbling and a single evaluator

who additively shares her input amongst the garblers. For $n > 3$, the correctness of GC can be concluded based on majority rule on the GCs received from the garblers.

4.4 4PC with god

In this section, we propose an efficient 5-round 4PC secure against one active corruption, assuming pairwise channels. Our protocol involves communication and computation of just one GC, in contrast to the protocol of [129] that requires 12 GCs. We take the route of employing two garblers and one evaluator as in our fair 3PC protocol. The fourth party simply shares its input amongst the rest. When the evaluator is honest, our protocol ensures that either an honest party identifies the corrupt party or a conflict (assured to include the corrupt party), or the honest evaluator is successful in GC evaluation by the end of Round 2. In the former case, the honest party would identify at least one honest party, to whom she sends her possessed input shares in Round 3. We use replicated secret sharing (RSS) that allows reconstruction of the output based on views of any two (honest) parties. In the latter scenario, the encoded output obtained upon GC evaluation is instantly used for output computation by all the parties in Round 3. Thus, in either scenario, at least one of the honest parties will be able to compute the output latest by Round 3 and everyone will receive it by Round 4. On the other hand, a corrupt evaluator can drag the honest parties up to Round 4 to reveal its identity. This is the only case that makes our protocol run for 5 rounds where the last round is used by the honest parties to exchange their possessed shares to compute the output on clear.

With the above high level idea, we describe a sub-protocol that enforces input consistency as per RSS and then we present our 5-round protocol $\mathsf{g4PC}$. Each party P_i ($i \in [4]$) maintains a pair of global sets– a corrupt set $\mathcal{C}_i$ and a conflict set $\mathcal{F}_i$ which respectively hold identities of the party detected to be corrupt and pairs of parties detected to be in conflict.

4.4.1 Protocol for Input Consistency

Our protocol $\mathsf{InputCommit}_i$ that runs for two rounds, enforces input consistency of party P_i's secret x_i as per RSS. Recall that as per RSS for three shareholders, P_i makes three shares of its secret x_i as $x_i = \oplus_{P_j \in \mathcal{P}_i} x_{ij}$ where $\mathcal{P}_i = [4] \setminus i$ denotes the shareholders (i.e. all but P_i). The share x_{ij} goes to all but P_i and P_j, namely to the set of parties in $\mathcal{P}_{ij} = \mathcal{P} \setminus \{P_i, P_j\}$. Now to ensure that a corrupt P_i remains committed to its secret or a corrupt shareholder P_j later cannot open a share of P_i differently, we use commitments on the shares. Namely, in the first round, commitments on input shares are distributed by P_i to all while the openings are sent only to the relevant shareholders. In the second round, the shareholders exchange the commitments received in the first round, while the openings are exchanged only with the

relevant shareholders. A simple majority rule suffices to conclude on the commitment c_{ij} of the 'committed' share x_{ij}. When no honest majority is found, it can be concluded that P_i is corrupt and his input is taken as a default value by all parties. When the commitment and the opening distributed by P_i is found to be inconsistent, then P_i is identified as corrupt. When the commitment as distributed by P_i and forwarded by P_j contradict, then P_i and P_j are put in conflict set.

A share x_{ij} is said to be 'committed' if each honest $P_\alpha \in \mathcal{P}_i$ holds c_{ij} and each honest $P_\beta \in \mathcal{P}_{ij}$ holds o_{ij} such that c_{ij} opens to x_{ij} via o_{ij}. A secret x_i is said to be 'committed' if each of its three shares are committed. An honest party always 'commits' to its secret. When a corrupt party does not commit to a secret, it is either identified as corrupt or found to be in conflict by at least one honest party. For the commitments, we use a strong NICOM according to which binding holds even for adversarially chosen public parameter of the NICOM (Section 2.4.2). Looking ahead the strong NICOM ensures that P_i itself cannot change its committed secret later and also cannot keep two different parties on different pages in terms of the opening information o_{ij}. Protocol $\mathsf{InputCommit}_i$ appears in Figure 4.2.

Protocol $\mathsf{InputCommit}_i()$

Inputs: Party P_i has x_i and others have no input.

Notation: $\mathcal{P}_i$ and $\mathcal{P}_{ij}$ denote the set $\mathcal{P} \setminus P_i$ and respectively $\mathcal{P} \setminus \{P_i, P_j\}$. $\mathsf{ind}(S)$ denotes the set of indices belonging to the parties in a set S.

Output: Each $P_k \in \mathcal{P}_i$ outputs $(\{\mathsf{c}_{ij}\}_{j\in\mathsf{ind}(\mathcal{P}_i)}, \{\mathsf{o}_{ij}, x_{ij}\}_{j\in\mathsf{ind}(\mathcal{P}_{ik})}, \mathcal{C}_k, \mathcal{F}_k)$. $\{\mathsf{c}_{ij}, \mathsf{o}_{ij}\}$ denote the commitment and opening of the share x_{ij}. $\mathcal{C}_k$ and $\mathcal{F}_k$ denote the corrupt and conflict set respectively.

Primitives: A NICOM $(\mathsf{sCom}, \mathsf{sOpen})$ with strong binding property (Section 2.4.2), a 3-party 1-private RSS (Section 4.2).

Round 1:

– P_i shares his input as $x_i = \oplus_{j\in\mathsf{ind}(\mathcal{P}_i)} x_{ij}$.

– P_i samples pp_i and generates commitments on shares x_{ij} for $j \in \mathsf{ind}(\mathcal{P}_i)$ as $(\mathsf{c}_{ij}, \mathsf{o}_{ij}) \leftarrow \mathsf{sCom}(\mathsf{pp}_i, x_{ij})$

– For every x_{ij}, P_i sends $(\mathsf{pp}_i, \mathsf{c}_{ij})$ to $\mathcal{P}_i$ and o_{ij} to $\mathcal{P}_{ij}$.

Round 2: With respect to every share x_{ij}, every P_k in $\mathcal{P}_{ij}$ sets $\mathcal{C}_k = \{P_i\}$ if $\mathsf{sOpen}(\mathsf{pp}_i, \mathsf{c}_{ij}, \mathsf{o}_{ij}) = \perp$. Otherwise, P_k forwards $(\mathsf{pp}_i, \mathsf{c}_{ij})$ to $\mathcal{P}_i$ and o_{ij} to $\mathcal{P}_{ij}$. Now P_k does the following local computation.

- Set $\mathcal{C}_k = \{P_l\}$ if P_l forwards an invalid opening i.e $\mathsf{sOpen}(\mathsf{pp}_i, \mathsf{c}_{ij}, \mathsf{o}_{ij}) = \perp$ holds for $(\mathsf{pp}_i, \mathsf{c}_{ij}, \mathsf{o}_{ij})$ sent by P_l.
- Set $\mathcal{F}_k = \{P_i, P_l\}$ if $(\mathsf{pp}_i, \mathsf{c}_{ij})$ received from P_i and forwarded by P_l do not match.
- Set $\mathcal{C}_k = \{P_i\}$, if there is no majority among the versions of $(\mathsf{pp}_i, \mathsf{c}_{ij})$ forwarded by the parties in $\mathcal{P}_i$. If $P_k \in \mathcal{P}_{ij}$, set x_{ij} to a default value (and commitments are assumed appropriately). Otherwise, set $(\mathsf{pp}_i, \mathsf{c}_{ij})$ as the majority value, o_{ij} as the corresponding opening, and $x_{ij} = \mathsf{sOpen}(\mathsf{pp}_i, \mathsf{c}_{ij}, \mathsf{o}_{ij})$.

Figure 4.2: Protocol $\mathsf{InputCommit}_i()$

Lemma 4.1 *If P_i is honest, its chosen input x_i is committed in $\mathsf{InputCommit}_i$.*

Proof: Since the corrupt party forms a minority in $\mathcal{P}_i$, irrespective of its behaviour in Round 2, every x_{ij} and therefore x_i remains committed. □

Lemma 4.2 *When corrupt P_i misbehaves, it belongs to either $\mathcal{C}_j$ or $\mathcal{F}_j$ of some honest P_j by the end of $\mathsf{InputCommit}_i$.*

Proof: For the jth ($j \in \mathsf{ind}(\mathcal{P}_i)$) share of P_i, it can misbehave in the following ways: (a) P_i sends different versions of $(\mathsf{pp}_i, \mathsf{c}_{ij})$ to the parties in $\mathcal{P}_i$; (b) P_i sends invalid opening o_{ij} (or does not send any opening) to some party in $\mathcal{P}_{ij}$. In the former case, all the honest parties will populate their corrupt set if there is no majority in P_i's commitments else they populate their conflict set with a pair, consisting of P_i. In the latter case, the honest recipient of the invalid opening will include P_i in its corrupt set. So the lemma holds. □

Lemma 4.3 *Either corrupt P_i 'commits' to an input or all honest parties agree on a default value by the end of $\mathsf{InputCommit}_i$.*

Proof: For the jth ($j \in \mathsf{ind}(\mathcal{P}_i)$) share of P_i, there are two cases based on whether P_i sends the same common message $(\mathsf{pp}_i, \mathsf{c}_{ij})$ to at least two among the parties in $\mathcal{P}_i$ with valid corresponding opening o_{ij} sent to every party in $\mathcal{P}_{ij}$. If not, the exchange of messages among the honest parties in Round 2 will not constitute a majority and all the honest parties would detect P_i to be corrupt and a default value will be taken as x_{ij}. Else, c_{ij} would be accepted as

the commitment for the jth share. The exchange of opening o_{ij} among the parties in $\mathcal{P}_{ij}$ ensure that they have access to the corresponding unique committed share x_{ij}. The uniqueness of the share is ensured by the binding property of commitment scheme. □

4.4.2 Our protocol

Without loss of generality, P_1, P_2 take the role of garblers and P_3 enacts the role of evaluator in our protocol g4PC. In parallel to running the input commitment sub-protocol for every party P_i, protocol g4PC, in similar spirit to our previous protocols, proceeds by having the garblers P_1 and P_2 share and utilize common randomness to compute individually the same garbled circuit and permuted commitments of the encoding information corresponding to the three shares of the inputs of all the parties. The permutation strings are used for all the shares for the sake of uniformity. Then the strings corresponding to the shares possessed by an evaluator are simply disclosed to her, emulating the case in the three-party protocols where no permutation string is needed for the shares of an evaluator to protect them from a bad garbler. As per RSS, a party P_α would ideally hold the shares $\{x_{ij}\}_{i\in[4],j\in\mathsf{ind}(\mathcal{P}_{i\alpha})}$ that include its three shares $\{x_{\alpha j}\}_{j\in\mathcal{P}_\alpha}$ and the two designated shares $\{x_{ij}\}_{j\in\mathcal{P}_{i\alpha}}$ of every other party P_i by the end of Round 1. Note that the latter shares may not be the committed ones and final committed values may differ by the end of Round 2 (say, if the majority turns out to be different or if a default value is assumed).

In the second round, while the garblers send the GCs, committed encoding information in permuted order, the relevant permutation strings on clear, the opening of the shares held by it, an evaluator checks the sanity of the received information, often leveraging the fact that at least one of the garblers is honest and would have computed the information correctly. The round-saving trick of composing the input commitment with the release of the encoded inputs for the shares in parallel leads to release of encoded inputs for non-committed shares, which in turn results in evaluation of the circuit on non-committed inputs. Evaluating the circuit only when no corrupt and no conflict is detected by the end of Round 2 would solve the problem for an honest evaluator, as this ensures encoded input for committed shares alone has been dealt. The trick to prevent a corrupt evaluator from getting output on non-committed inputs is to withhold (yet commit in Round 1) the decoding information for an oblivious garbling scheme and release the (hash of) decoding information only upon a confirmation that an encoded output is computed using committed inputs. The simple check that a corrupt evaluator has no conflict with any of the garblers ensures that the garblers must be in possession of the committed shares of the corrupt evaluator by the end of first round itself and so the released encoded inputs correspond to the committed shares (and the encoded output corresponds to committed inputs).

The repetitive disbursal of shares in RSS brings along another issue. Both the garblers possess the share x_{34}. An evaluator receives encoded input for these shares from both the garblers, as per the protocol. A corrupt evaluator P_3 can exploit this step to obtain encoded inputs for two different versions of the share x_{34} (by dealing to the garblers) and subsequently evaluates the circuit on multiple inputs. While having the decoding information hidden would not leak the clear outputs, the corrupt evaluator, on holding the the encoded outputs, can conclude if its two different chosen inputs lead to the same output or not. While the issue is very subtle, the fix is quite easy where only one *pre-determined* garbler is given responsibility of releasing the encoded input for the common share x_{34}. In order to avoid repeated disclosing of encoded outputs of the common shares between the garblers, this approach is taken for all the common shares, namely $\{x_{13}, x_{14}, x_{23}, x_{24}, x_{34}, x_{43}\}$. To balance load, we ask P_1 to open encoded inputs for $\{x_{13}, x_{14}, x_{34}\}$ and P_2 to take care of the rest.

In Round 3, if any party identifies the corrupt or any conflict, it sends the openings for all the shares that it possesses from the input commitment protocol to a party who remains outside the corrupt and conflict sets and thus guaranteed to be honest. This special party is denoted as TTP who takes care of reconstructing all the inputs and computing the output on clear and lastly handing it over to all the parties in the next round. Even a corrupt evaluator cannot make an honest TTP to compute an output on anything other than committed inputs. The strong binding property of the commitments does not allow a corrupt evaluator to change its *own* committed shares. To disambiguate about the identity of TTP, a party when disclosing its opening to its selected TTP notifies the identity of the designated TTP to all. When a TTP takes responsibility, all the parties safely accept the output relayed by the TTP in the next round, for a TTP is never corrupt. An honest party will never elect a corrupt party as a TTP and a corrupt evaluator does not have a corrupt companion to enact a TTP. Therefore, if an honest party elects a TTP in Round 3, all terminate the protocol with output by Round 4. On the other hand when no conflict and no corrupt is detected, an honest evaluator computes the encoded output and forwards the same to the garblers in Round 3. Similarly, an honest garbler opens the (hash of) decoding information to P_3 and P_4. We use preimage-resistant hash to enable P_3 and P_4 to compute the output while preserving the authenticity of the garbling scheme. For an honest evaluator, then all parties compute the output by the end of Round 3 itself via the encoded output and decoding information. A corrupt evaluator, however, can keep the honest parties on different pages in terms of the identity of TTP, while not disclosing its possessed shares to anyone. In this case, the honest parties realize that the evaluator P_3 is corrupt earliest at the end of Round 4. They can then exchange their shares in Round 5 to compute the output on clear like a TTP does. The protocol appears in Figure 4.3.

Protocol g4PC()

Inputs: Party P_α has x_α for $\alpha \in [4]$.

Common Inputs: The circuit $C(x_1, x_2, x_3, x_4)$ that computes $f(x_{12} \oplus x_{13} \oplus x_{14}, x_{21} \oplus x_{23} \oplus x_{24}, x_{31} \oplus x_{32} \oplus x_{34}, x_{41} \oplus x_{42} \oplus x_{43})$ each input, their shares and output are from $\{0,1\}^\ell$. P_3 is the evaluator and (P_1, P_2) are the garblers.

Output: $y = C(x_1, x_2, x_3, x_4)$

Primitives: $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ that is correct, private, oblivious and authentic, a NICOM $(\mathsf{Com}, \mathsf{Open})$, a PRG G, a preimage-resistant Hash H and sub-protocol $\mathsf{InputCommit}_\alpha$ (Figure 4.2) for every $P_\alpha \in \mathcal{P}$.

Round 1: Round 1 of $\mathsf{InputCommit}_\alpha()$ for every $P_\alpha \in \mathcal{P}$ is run. In parallel,

- P_1 chooses random seed $s \in_R \{0,1\}^\kappa$ for G and sends s to P_2.
- P_3 samples pp_3 for NICOM and sends to P_1, P_2.

Round 2: Round 2 of $\mathsf{InputCommit}_\alpha()$ is run. In parallel,

- $P_g (g \in [2])$ locally computes the following:
  - Compute garbled circuit $(\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C)$ using randomness from $\mathsf{G}(s)$. Assume $\{\mathsf{K}^0_\alpha, \mathsf{K}^1_\alpha\}_{\alpha \in [3\ell]}$, $\{\mathsf{K}^0_{3\ell+\alpha}, \mathsf{K}^1_{3\ell+\alpha}\}_{\alpha \in [3\ell]}$, $\{\mathsf{K}^0_{6\ell+\alpha}, \mathsf{K}^1_{6\ell+\alpha}\}_{\alpha \in [3\ell]}$, $\{\mathsf{K}^0_{9\ell+\alpha}, \mathsf{K}^1_{9\ell+\alpha}\}_{\alpha \in [3\ell]}$ correspond to the encoding information for the input shares of P_1, P_2, P_3, P_4 respectively (w.l.o.g).
  - Let $p_{ij} \in_R \{0,1\}^\ell$ be permutation string for input wires derived from randomness $\mathsf{G}(s)$ corresponding to P_i's shares i.e $\{x_{ij}\}_{j \in \mathsf{ind}(\mathcal{P}_i)}$ for $i \in [4]$ and $p_i \leftarrow ||_{j \in \mathsf{ind}(\mathcal{P}_i)} p_{ij}$.
  - Generate commitments to e and d using randomness from $\mathsf{G}(s)$. For $b \in \{0,1\}$ and $\alpha \in [3\ell]$, compute $(c^b_\alpha, \mathrm{o}^b_\alpha) \leftarrow \mathsf{Com}(\mathsf{pp}_3, \mathsf{e}^{p^\alpha_1 \oplus b}_\alpha)$, $(c^b_{3\ell+\alpha}, \mathrm{o}^b_{3\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, \mathsf{e}^{p^\alpha_2 \oplus b}_{3\ell+\alpha})$, $(c^b_{6\ell+\alpha}, \mathrm{o}^b_{6\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, \mathsf{e}^{p^\alpha_3 \oplus b}_{6\ell+\alpha})$, $(c^b_{9\ell+\alpha}, \mathrm{o}^b_{9\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, \mathsf{e}^{p^\alpha_4 \oplus b}_{9\ell+\alpha})$. Let $(\mathsf{c}, \mathsf{o}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, \mathsf{H}(\mathsf{d}))$. Set $\mathcal{B} = \big\{\mathsf{C}, \{c^b_\alpha\}_{\alpha \in [12\ell], b \in \{0,1\}}, c, \{p_{ij}\}_{i \in [4], j \in \mathsf{ind}(\mathcal{P}_{i3})}\big\}$, where $\{p_{ij}\}_{i \in [4], j \in \mathsf{ind}(\mathcal{P}_{i3})}$ refer to the permutation strings of wires corresponding to the shares known to P_3.
- $P_g (g \in [2])$ sends $\mathcal{B}$ to P_3 and c to P_4. If $\mathcal{C}_g = \emptyset$, P_g sends the openings of the commitments in $\mathcal{B}$ corresponding to $\{x_{ij}\}_{i \in [4], j \in \mathsf{ind}(\mathcal{P}_{ig})}$ i.e the input shares that it holds at end of **Round 1** and $\mathsf{M}_g = \{m_{ij}\}_{i \in [4], j \in \mathsf{ind}(\mathcal{P}_{ig})}$ where $m_{\alpha\beta} = p_{\alpha\beta} \oplus x_{\alpha\beta}$. The common shares, however, are opened by one garbler. The openings corresponding to commitment of $\{x_{13}, x_{14}, x_{34}\}$ are sent only by P_1. The openings corresponding to commitment of $\{x_{23}, x_{24}, x_{43}\}$ are sent only by P_2.
- P_3 locally does

- Add $\{P_1, P_2\}$ to $\mathcal{F}_3$ if $\mathcal{B}$ received from P_1, P_2 is not identical.
- If $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$ (indicating no conflict with the garblers so far), then (a) add P_g to $\mathcal{C}_3$ ($g \in [2]$) when the indices $\{\bar{m}_{ij} = p_{ij} \oplus x_{ij}\}_{i\in[4], j\in \mathsf{ind}(\mathcal{P}_{i3})}$, computed using its version of x_{ij} and p_{ij}, received from P_g, mismatches with $\{m_{ij}\}_{i\in[4], j\in \mathsf{ind}(\mathcal{P}_{i3})}$ received from P_g; (b) add (P_1, P_2) to $\mathcal{F}_3$ when $\mathsf{M}_1, \mathsf{M}_2$ received from them is not consistent w.r.t. $\{m_{13}, m_{14}, m_{23}, m_{24}, m_{34}, m_{43}\}$.
- If $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$, then add P_g to $\mathcal{C}_3$ when any of the openings sent by P_g ($g \in [2]$) results to $\perp$. Otherwise, it sets $\mathsf{X} = ||_{i\in[4], j\in \mathsf{ind}(\mathcal{P}_i)}\mathsf{X}_{ij}$, where X_{ij} contains encoded input for x_{ij} and computes $\mathsf{Y} \leftarrow \mathsf{Ev}(\mathsf{C}, \mathsf{X})$ with $\mathsf{C} \in \mathcal{B}$.

– P_4 locally adds $\{P_1, P_2\}$ to $\mathcal{F}_4$ if c received from them do not match.

Round 3:

– If $\mathcal{C}_\alpha \neq \emptyset \vee \mathcal{F}_\alpha \neq \emptyset$, P_α ($\alpha \in [4]$) sends $\mathsf{V}_\alpha = \{\mathsf{o}_{ij}\}_{i\in[4], j\in \mathsf{ind}(\mathcal{P}_{i\alpha})}$ to P_β where $P_\beta \notin \mathcal{C}_\alpha \cup \mathcal{F}_\alpha$ and (TTP, β) to all.

– If $\mathcal{C}_g = \mathcal{F}_g = \emptyset$, P_g ($g \in [2]$) sends o to P_3, P_4.

– If $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$, P_3 sends Y to P_1, P_2 and P_4.

– If P_α ($\alpha \in [4]$) receives V_β from P_β in Round 3, it uses V_β to open its missing shares $\{x_{i\alpha}\}_{i\in[4]\setminus\{\alpha\}}$. If one of the opening leads to $\perp$, set $\mathcal{C}_\alpha = P_\beta$. Else compute $y = f(\oplus_{j\in \mathsf{ind}(\mathcal{P}_1)}x_{1j}, \oplus_{\mathsf{ind}(\mathcal{P}_2)}x_{2j}, \oplus_{\mathsf{ind}(\mathcal{P}_3)}x_{3j}, \oplus_{\mathsf{ind}(\mathcal{P}_4)}x_{4j})$.

– If P_g ($g \in [2]$) receives Y from P_3 such that $P_3 \notin \mathcal{C}_g$ and $(P_3, P_1), (P_3, P_2) \notin \mathcal{F}_g$, then compute $y \leftarrow \mathsf{De}(\mathsf{Y}, \mathsf{d})$. If P_4 receives Y as above and o from one of the P_gs, it computes y after recovering $\mathsf{H}(\mathsf{d}) \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{c}, \mathsf{o})$. If $P_1/P_2/P_4$ receives invalid Y, they populate their respective corrupt set $\mathcal{C}$ with P_3. If P_3 receives o, then it computes $\mathsf{H}(\mathsf{d})$ and subsequently y.

Round 4:

– If P_α computed y, it sends (y, TTP) when elected as TTP and y otherwise to all and terminates.

– If (TTP, β) is received in Round 3 and (y, TTP) is received from P_β, a party P_α outputs y and terminates. If only the former condition is true, then P_α identifies the sender of the message (TTP, β) as corrupt.

– If $\mathcal{C}_\alpha \neq \emptyset$ and y is received from a party not in $\mathcal{C}_\alpha$, P_α outputs y and terminate.

Round 5: If P_α ($\alpha \in [4]$) has not terminated yet, it sends its view V_α to every party in $\mathcal{P} \setminus \mathcal{C}_\alpha$. On receiving V_β from some $P_\beta \notin \mathcal{C}_\alpha$, it computes y as a TTP does and terminates.

Figure 4.3: Protocol $\mathsf{g4PC}()$

4.4.3 Correctness and Security

We prove the correctness via a sequence of lemmas.

Lemma 4.4 *For honest P_i, P_j, $P_i \notin \mathcal{C}_j$ holds.*

Proof: An honest P_j would add P_i to $\mathcal{C}_j$ if one of the following are true: (a) During $\mathsf{InputCommit}_i$ if either there is no majority among the version of $(\mathsf{pp}_i, \mathsf{c}_{ij})$ received from the set of parties $\mathcal{P}_i$ or P_j receives an invalid opening corresponding to commitment on input share from P_i; (b) garbler P_i sends labels inconsistent with the message that it sent to evaluator P_j in Round 1; (c) garbler P_i's opening of committed encoded input of GC sent to evaluator P_j fails; (d) evaluator P_i sends an invalid Y to P_j; (e) P_i assigns P_j to be the TTP and sends V_i comprising of invalid openings of committed shares; (f) P_j received (TTP, β) from P_i but no output is received from P_β in Round 4. Since none of the above can occur for honest P_i and P_j, the lemma holds. □

Lemma 4.5 *A pair of honest parties cannot belong to $\mathcal{F}_i$ of an honest P_i.*

Proof: An honest P_i would add (P_j, P_k) to $\mathcal{F}_i$ if one of the following holds: (a) During execution of $\mathsf{InputCommit}_j$, the versions of P_j's commitment on its input shares received by P_i from P_j and P_k were inconsistent (analogous condition w.r.t. $\mathsf{InputCommit}_k$); (b) when (P_j, P_k) are garblers, $P_i = P_4$ and o received from P_j, P_k is not identical; (c) (P_j, P_k) are garblers, $P_i = P_3$ and: (c.1) $\mathcal{B}$ received from P_j, P_k is not identical (c.2) when $\mathcal{F}_i = \emptyset$ at the end of of all the four executions of $\mathsf{InputCommit}$ but the indices received by P_i from the garblers corresponding to the common shares held by them do not match i.e when $\mathsf{M}_j, \mathsf{M}_k$ received from them is not consistent. It is easy to verify that cases (a), (b) and (c.1) cannot occur for honest P_j, P_k. Regarding case (c.2), the argument follows from the fact that P_j, P_k must be in agreement with respect to corrupt party's (say P_l) input shares at the end of Round 1 itself. If not, then the version forwarded by at most one among (P_j, P_k) (say P_j) during $\mathsf{InputCommit}_l$ can match with the one P_i received by P_l, leading to P_i populating $\mathcal{F}_i$ with $\{P_l, P_k\}$. This contradicts the assumption in case (c.2) regarding $\mathcal{F}_i = \emptyset$ at the end of of all executions of $\mathsf{InputCommit}$; completing the proof. □

Lemma 4.6 *The encoded output Y computed by an honest P_3 corresponds to the committed inputs of all parties.*

Proof: An honest P_3 evaluates the GC and computes Y when both $\mathcal{F}_3$ and $\mathcal{C}_3$ are empty. This implies that the corrupt party 'commits' to its input in Round 1 of its $\mathsf{InputCommit}$ instance (by Lemma 4.2). We can thus conclude that the honest garbler would possess committed input shares of all parties at the end of Round 1 itself and open the encoded inputs accordingly. A potentially corrupt garbler is forced to send the encoded inputs corresponding to committed inputs. Because– (a) if corrupt garbler tries to open different encoded inputs for the shares known to P_3, then he is added to $\mathcal{C}_3$; (b) if it tries to open different encoded inputs for the shares not known to P_3, then P_3 would add the pair of garblers to $\mathcal{F}_3$. Thus, in either case, P_3 does not evaluate as at least one among $\mathcal{F}_3$, $\mathcal{C}_3$ is non-empty. □

Lemma 4.7 *If the encoded output* Y *of a corrupt evaluator* P_3 *is used for output computation by an honest garbler, then it must correspond to committed inputs of all parties.*

Proof: An honest garbler, say P_g releases the opening information o for $\mathsf{H}(d)$ and uses the encoded output Y (such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) \neq \perp$) received from evaluator P_3 to compute output if $P_3 \notin \mathcal{C}_g$ and $(P_3, P_1), (P_3, P_2) \notin \mathcal{F}_g$. Lemma 4.2 implies that P_3 did not misbehave in $\mathsf{InputCommit}_3$ at all and has committed a unique input in Round 1. This implies that P_3 receives encoded inputs for committed shares and authenticity ensures that Y corresponds to the committed inputs of all the parties. Note that authenticity of the garbling scheme is preserved since P_3 receives only the preimage-resistant hash of the decoding information in the form $\mathsf{H}(\mathsf{Y}_0)||\mathsf{H}(\mathsf{Y}_1)$ corresponding to each output wire (enabling P_3 to compute the output). Here, $\mathsf{Y}_0, \mathsf{Y}_1$ refer to the labels for values 0 and 1 respectively corresponding to an output wire. □

Lemma 4.8 *Protocol* $\mathsf{g4PC}$ *is correct.*

Proof: We argue that the output y computed corresponds to the unique inputs committed by each P_i ($i \in [4]$) during $\mathsf{InputCommit}_i$. It follows from Lemmas 4.3, 4.1 respectively that a corrupt party is forced to commit to a unique input and the honest parties' inputs are established as the committed inputs with public commitments by the end of parallel executions of $\mathsf{InputCommit}$. According to the protocol, an honest party P_α computes output in one of the following ways: (a) via decoding the encoded output Y; (b) via the V_β received from P_β on being elected as TTP; (c) on receiving y from an honest party; (d) on receiving (y, TTP) from P_β and (TTP, β) from some other party. In case (a), irrespective of whether P_3 is honest or corrupt, correctness follows from Lemma 4.6–4.7. The strong binding property of commitment scheme implies the output computed in case (b) is correct irrespective of whether P_β is honest

or corrupt. The correctness for case (c) follows from case (a) and the fact that the message was received from an honest party. The last case is argued as follows. The chosen TTP, P_β, is honest, irrespective of whether the message (TTP, β) is received from a corrupt or an honest party. While the former follows from the fact that a corrupt party does not have a corrupt companion to elect, the latter follows from Lemma 4.4–4.5. Now the correctness follows in case (d) from case (b). □

While the full proof of security appears in Section 4.8.2, we provide intuition for guaranteed output delivery and state the theorem below. If the corrupt party misbehaves in one of the InputCommit instances or while communicating the GC and openings on commitment of input labels (as a garbler in round 2), then an honest party invokes TTP on identifying the corrupt or detecting a conflict in Round 3. All the parties get output in Round 4. Otherwise, if P_3 is honest and gives out Y, then all the honest parties compute output by the end of Round 3 itself using hash of the decoding information sent by one of the garblers and Y. A corrupt P_3 can neither receive decoding information for his non-committed input nor convince honest parties about the corresponding Y. If Y corresponds to its committed input but it sends it only to some honest party or none, the remaining honest parties will receive output from the honest party who receives Y or through V_βs sent by other honest parties in Round 5.

Theorem 4.3 *Assuming one-way permutations, protocol* g4PC *securely realizes the functionality* $\mathcal{F}_{\mathsf{god}}$ *(Figure 2.4) against a malicious adversary that corrupts at most one party.*

4.4.4 Optimizations

The communication efficiency of our g4PC can be boosted similar to as described for f3PC in Section 4.3.2.

4.5 4PC with god in four rounds

In this section, we propose an efficient 4-round 4-party protocol secure against one active corruption, assuming pairwise channels. Deviating from the approach of [129, 159] and our proposals for 3PC and 4PC, we explore the setting of multiple evaluators, namely two evaluators and two garblers. With a guarantee of an honest evaluator, this protocol achieves guaranteed output delivery at the expense of communication and computation of two copies of the same GC.

The protocol ensures that the honest evaluator is either successful in GC evaluation or some honest party identifies a corrupt party or a pair of parties in conflict (assured to include the corrupt party) by the end of Round 2. In the former case, the encoded output obtained upon

GC evaluation is used for output computation in Round 3 itself. In the latter case, the honest party, having identified at least one honest party, sends his possessed input shares in Round 3. The use of replicated secret sharing (RSS) allows reconstruction of the output based on views of two honest parties by the end of Round 3. All parties obtain output by the end of Round 4.

The single evaluator and three garblers approach seems to require a minimum of 5 rounds (when the evaluator is corrupt) while requiring the same amount of communication. With the above high level idea, we proceed to present our protocol. We reuse the protocol for input consistency (Figure 4.2). Similar to our g4PC protocol, each party P_i ($i \in [4]$) maintains a pair of global sets– a corrupt set $\mathcal{C}_i$ and a conflict set $\mathcal{F}_i$ which respectively hold identities of the party detected to be corrupt and pairs of parties detected to be in conflict.

4.5.1 Our protocol

Without loss of generality, P_1, P_2 take the role of garblers and P_3, P_4 enact the role of evaluators in our protocol g4PC4. We reuse most of the tricks from our 5-round protocol and leverage the presence of an honest evaluator. Specifically, the corrupt evaluator, unlike in our 5-round protocol, cannot drag all the honest parties all the way to Round 4 for its detection. If everything goes as per the protocol and so no honest party elects a TTP in the end of Round 2, the honest evaluator must be able to compute the encoded output Y by the end of Round 2 and help all to get the output in Round 3. Otherwise, all the parties get output via a TTP by Round 4. The presence of an additional evaluator needs communicating one extra copy of the GC. We present the protocol g4PC4 in Figure 4.4

Protocol g4PC4()

Inputs: Party P_α has x_α for $\alpha \in [4]$.

Common Inputs: The circuit $C(x_1, x_2, x_3, x_4)$ that computes $f(x_{12} \oplus x_{13} \oplus x_{14}, x_{21} \oplus x_{23} \oplus x_{24}, x_{31} \oplus x_{32} \oplus x_{34}, x_{41} \oplus x_{42} \oplus x_{43})$ each input, their shares and output are from $\{0,1\}^\ell$. P_3, P_4 are the evaluators and (P_1, P_2) are the garblers.

Output: $y = C(x_1, x_2, x_3, x_4)$

Primitives: $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ that is correct, private, oblivious and authentic, a NICOM (Com, Open) a PRG G, a 3-party 1-private RSS, pre-image resistant Hash H and sub-protocol $\mathsf{InputCommit}_\alpha$ (Figure 4.2) for every $P_\alpha \in \mathcal{P}$.

Round 1: Round 1 of $\mathsf{InputCommit}_\alpha$ for every $P_\alpha \in \mathcal{P}$ is run. In parallel,

– P_1 chooses random seed $s \in_R \{0,1\}^\kappa$ for G and sends s to P_2.

– P_v ($v \in \{3,4\}$) samples pp_v for NICOM and sends to P_1, P_2.

Round 2: Round 2 of $\mathsf{InputCommit}_\alpha$ is run. In parallel,

– $P_g(g \in [2])$ locally computes $\mathcal{B}_3$ exactly the way $\mathcal{B}$ is computed in Protocol g4PC. It also computes $\mathcal{B}_4$ with respect to pp_4 in a similar way.

– $P_g(g \in [2])$ sends $\mathcal{B}_3$ to P_3. If $\mathcal{C}_g = \emptyset$, P_g sends the openings of the commitments in $\mathcal{B}_3$ corresponding to $\{x_{ij}\}_{i\in[4],j\in\mathsf{ind}(\mathcal{P}_{ig})}$ i.e the input shares that it holds at end of **Round 1** and $\mathsf{M}_g = \{m_{ij}\}_{i\in[4],j\in\mathsf{ind}(\mathcal{P}_{ig})}$ where $m_{\alpha\beta} = p_{\alpha\beta} \oplus x_{\alpha\beta}$. Analogous steps are executed with respect to P_4. The common shares, however, are opened by one garbler. The openings corresponding to commitment of $\{x_{13}, x_{14}, x_{34}\}$ are sent only by P_1. The openings corresponding to commitment of $\{x_{23}, x_{24}, x_{43}\}$ are sent only by P_2.

– P_v ($v \in \{3,4\}$)) local computation step is same as that of P_3 in g4PC (with respect to $\mathcal{C}_v$ and $\mathcal{F}_v$).

Round 3:

– If $\mathcal{C}_\alpha \neq \emptyset \vee \mathcal{F}_\alpha \neq \emptyset$, P_α ($\alpha \in [4]$) sends $\mathsf{V}_\alpha = \{\mathsf{o}_{ij}\}_{i\in[4],j\in\mathsf{ind}(\mathcal{P}_{i\alpha})}$ to P_β where $P_\beta \notin \mathcal{C}_\alpha \cup \mathcal{F}_\alpha$ and (TTP, β) to all.

– If $\mathcal{C}_g = \mathcal{F}_g = \emptyset$, P_g ($g \in [2]$) sends o to P_3, P_4.

– If $\mathcal{C}_v = \mathcal{F}_v = \emptyset$, P_v ($v \in \{3,4\}$) sends Y to all.

– If P_α ($\alpha \in [4]$) receives V_β from P_β in Round 3, it uses V_β to open its missing shares $\{x_{i\alpha}\}_{i\in[4]\setminus\{\alpha\}}$. If one of the opening leads to $\perp$, set $\mathcal{C}_\alpha = P_\beta$. Else compute $y = f(\oplus_{j\in\mathsf{ind}(\mathcal{P}_1)}x_{1j}, \oplus_{\mathsf{ind}(\mathcal{P}_2)}x_{2j}, \oplus_{\mathsf{ind}(\mathcal{P}_3)}x_{3j}, \oplus_{\mathsf{ind}(\mathcal{P}_4)}x_{4j})$.

– If P_g ($g \in [2]$) receives a valid Y from P_v such that $P_v \notin \mathcal{C}_g$ and $(P_v, P_1), (P_v, P_2) \notin \mathcal{F}_g$, then compute $y \leftarrow \mathsf{De}(\mathsf{Y}, \mathsf{d})$. If P_v receives o from one of the P_gs, it computes y after recovering $\mathsf{H}(\mathsf{d}) \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{c}, \mathsf{o})$.

Round 4:

– If P_α computed y via being elected as TTP, it sends (y, TTP) to all and terminates.

– If (TTP, β) is received in Round 3 and (y, TTP) is received from P_β, a party P_α outputs y and terminates.

Figure 4.4: Protocol g4PC4()

4.5.2 Correctness and Security

The proof for correctness appear below.

Lemma 4.9 *For honest P_i, P_j, $P_i \notin \mathcal{C}_j$ holds.*

Proof: The proof follows directly from the Lemma 4.4. □

Lemma 4.10 *Consider honest P_i. A pair of honest parties cannot belong to $\mathcal{F}_i$.*

Proof: An honest P_i would add (P_j, P_k) to $\mathcal{F}_i$ if one of the following holds: (a) During execution of $\mathsf{InputCommit}_j$, the versions of P_j's commitment on its input shares received by P_i from P_j and P_k were inconsistent. (Analogous condition wrt $\mathsf{InputCommit}_k$) (b) When (P_j, P_k) are garblers, P_i is evaluator and: (b.1) $\mathcal{B}_i$ received from P_j, P_k is not identical (b.2) When $\mathcal{F}_i = \emptyset$ at the end of all executions of $\mathsf{InputCommit}_m (m \in [4])$ but the indices received by P_i from the garblers corresponding to the common shares held by them do not match i.e when $\mathsf{M}_j, \mathsf{M}_k$ received from them is not consistent. It is easy to verify that cases (a) and (b.1) cannot occur for honest P_j, P_k. For case (b.2), the argument follows from the fact that P_j, P_k must be in agreement with respect to corrupt party's (say P_l) input shares at the end of Round 1 itself. If not, then the version forwarded by atmost one among (P_j, P_k) (say P_j) during $\mathsf{InputCommit}_l$ could match the one P_i received by P_l, leading to P_i populating $\mathcal{F}_i$ with $\{P_l, P_k\}$. This contradicts the assumption in case (b.2) regarding $\mathcal{F}_i = \emptyset$ at the end of of all executions of $\mathsf{InputCommit}$; completing the proof. □

Lemma 4.11 *The encoded output Y computed by an honest evaluator corresponds to the committed inputs of all parties.*

Proof: Consider an honest evaluator P_i. If $i = 3$, the proof follows exactly as described in Lemma 4.6. Else if $i = 4$, the proof of Lemma 4.6 still holds, except in that $P_3, \mathcal{F}_3, \mathcal{C}_3$ are replaced with $P_4, \mathcal{F}_4, \mathcal{C}_4$. □

Lemma 4.12 *If the encoded output sent by a potentially corrupt evaluator is used for output computation by an honest garbler, it must correspond to committed inputs of all parties.*

Proof: Similar to our $\mathsf{g4PC}$ protocol, an honest garbler, say P_g uses the encoded output Y (such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) \neq \perp$) received from evaluator P_v to compute output only if $P_v \notin \mathcal{C}_g$ and $(P_v, P_1), (P_v, P_2) \notin \mathcal{F}_g$ at the end of round 2. Correspondingly, if $\mathcal{C}_g = \mathcal{F}_g = \emptyset$, P_g would also send o to both the evaluators in round 3. This ensures that Y corresponds to committed

inputs as follows: Although P_v may be corrupt, however, Lemma 4.2 implies that P_v did not misbehave in $\mathsf{InputCommit}_v$ at all and has committed a unique input in Round 1. As a result, P_v receives encoded inputs for committed shares and authenticity ensures that Y corresponds to the committed inputs of all the parties. Note that authenticity of the garbling scheme is preserved since P_v receives only the preimage-resistant hash of the decoding information. □

Theorem 4.4 *Protocol* g4PC4 *is correct.*

Proof: We argue that the output y computed corresponds to the unique inputs committed by each P_i ($i \in [4]$) during $\mathsf{InputCommit}_i$. It follows from Lemmas 4.1, 4.3 that a corrupt party is forced to commit to its input and the honest parties' inputs are established as the committed inputs with public commitments by the end of parallel executions of InputCommit. According to the protocol, output computation is done by one of the following cases: (a) by decoding the encoded output Y sent by an evaluator (b) by V_α received from P_α on being elected as a TTP. (c) by receiving (y, TTP) from a party P_β when (TTP, β) was received in round 3. Case (a) follows directly from Lemma 4.12 and 4.11. In case (b), since the TTP is honest, the strong binding property of commitments established by Round 2 ensures the correctness of output computed by the TTP, irrespective of whether P_α is honest or not. For case (c), the chosen TTP, P_β, is honest, irrespective of whether the message (TTP, β) is received from a corrupt or an honest party. While the former follows from the fact that a corrupt party does not have a corrupt companion to elect, the latter follows from Lemma 4.10 and 4.9. Now the correctness follows in case (c) from case (b). □

While the sketch of proof of security appears in Section 4.8.3 (the full proof and intuition for achieving guaranteed output delivery is similar to our 5-round 4PC), we state the theorem below.

Theorem 4.5 *Assuming one-way permutations, our protocol* g4PC4 *securely realizes the functionality* $\mathcal{F}_{\mathsf{god}}$ *(Figure 2.4) against a malicious adversary that can corrupt at most one party.*

4.5.3 Optimizations

The communication efficiency of our g4PC4 can be boosted similar to as described for f3PC in Section 4.3.2. Additionally, computation of commitment on encoding information by a garbler wrt pp (for NICOM) sent by each of the two evaluators can be avoided as follows: P_3 alone chooses pp used for commitment on encoding information and sends pp to all. The message from garblers can include pp, allowing P_4 to check if the garblers and P_3 are in agreement with respect to pp or populate the conflict set accordingly based on mismatch.

4.6 3PC with god

In this section, we describe our efficient 3PC protocol, g3PC with god. This protocol necessarily requires a broadcast channel [67]. In accordance with our goal of computation and communication efficiency, the broadcast communication complexity of our (optimized) protocol is independent of circuit size. In terms of communication over private channels, g3PC involves a single GC and is therefore comparable to [159].

Starting with the protocol of [159], the main idea of our protocol is centered around the following neat trick. In a situation where it is publicly known that a pair of parties is in conflict, it must be the case that one among the two specific parties is corrupt. It follows that the third party is honest and therefore entitled to act as the trusted-third party (TTP). Suppose such a TTP is established during the protocol, the other parties send their inputs on clear to this TTP who computes the function on direct inputs and forwards the output to all. Banking on this intuition, we now proceed to give a high-level description of our protocol.

In the first round, similar to f3PC, P_3 shares his input while the garblers agree upon common randomness. In round 2, garblers broadcast the common message computed using shared randomness, namely the GC and commitment on encoding information. Additionally, the garblers privately send the opening of relevant commitments, namely corresponding to their own input and the input share of P_3 held by them. If the broadcast messages are identical and openings are valid then P_3 can begin evaluating the GC. However, if the broadcast messages mismatch, it can be publicly inferred that P_1, P_2 are in conflict and therefore P_3 is eligible to enact the role of TTP. We extend this idea to the case when broadcast messages are identical but P_3 locally identifies one of the garblers to be corrupt. In this scenario, say P_3 identified P_2 to be corrupt. Then, P_3 makes this conflict public in Round 3 via broadcast. Consequently P_1 is entitled to act as the TTP. The protocol ensures that if P_3 fails to evaluate the GC, a TTP is established at most by Round 3. If the TTP is established, the parties send their inputs on clear to the TTP in Round 4 who computes and subsequently sends the output to all in the final round of the protocol.

An issue that surfaces in the above approach is that a corrupt P_3 who has successfully evaluated the GC with respect to his input x_3 shared in the round 1, might pretend to be in conflict with one of the garblers, say P_2. Now P_1 would be established as the TTP. P_3 can now send $x_3' \neq x_3$ to P_1 and get the output corresponding to x_3' as well. This violates security since P_3 gets outputs corresponding to his two chosen inputs. To handle this, we adopt the following strategy: The evaluator P_3 broadcasts the commitment on his shares in Round 1 and sends the openings of shares to the respective garbler. A garbler who receives invalid opening is

allowed to publicly raise a conflict with P_3 in Round 2, establishing his co-garbler as the TTP. If valid openings are issued, P_3 is committed to each of his shares and therefore his input. The binding property of commitment ensures that the TTP computes output with respect to P_3's shares distributed in Round 1. Tying up the loose ends, if P_3 is identified to be corrupt by both garblers, then P_1 is chosen to be the TTP by default.

In a nutshell, P_3 acts as TTP only when common message broadcast by garblers are not identical. Contrarily, a garbler, say P_1, is TTP when either P_3 locally identified P_2 to be corrupt at the end of Round 2 (due to invalid opening of commitment on encoded inputs) or P_2 found P_3 to be corrupt at the end of Round 1 (inconsistent opening of commitment of P_3's input share sent to P_2). Also, P_1 is chosen as TTP by default when both garblers identify P_3 to be corrupt. The formal description of the protocol appears in Figure 4.5 and its proofs appear below. Our proposed optimizations which are incorporated in our implementation are given below.

Protocol g3PC

Inputs: Party P_α has x_α for $\alpha \in [3]$.

Common Inputs: Same as f3PC.

Output: $y = C(x_1, x_2, x_3, x_4) = f(x_1, x_2, x_3 \oplus x_4)$

Primitives: A garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ that is correct, private and authentic with the property of soft decoding, a NICOM $(\mathsf{Com}, \mathsf{Open})$ and a PRG G.

Round 1:

– P_1 chooses random seed $s \in_R \{0,1\}^\kappa$ for G and sends s to P_2.

– P_3 picks $x_{31}, x_{32} \in_R \{0,1\}^\ell$ with $x_3 = x_{31} \oplus x_{32}$. P_3 samples pp for NICOM and generates $(c_{31}, \mathrm{o}_{31}) \leftarrow \mathsf{Com}(\mathsf{pp}, x_{31})$, $(c_{32}, \mathrm{o}_{32}) \leftarrow \mathsf{Com}(\mathsf{pp}, x_{32})$, broadcasts $\{\mathsf{pp}, c_{31}, c_{32}\}$ and sends $(x_{31}, \mathrm{o}_{31})$, $(x_{32}, \mathrm{o}_{32})$ to P_1, P_2 respectively.

Round 2:

– $P_i (i \in [2])$ broadcasts $(\mathtt{Conflict}, P_3)$ if $\mathsf{Open}(c_{3i}, \mathrm{o}_{3i}) \neq x_{3i}$. Else, it does the following:

  ◦ Compute GC $(\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C)$ using randomness from $G(s)$. Assume $\{\mathsf{K}^0_\alpha, \mathsf{K}^1_\alpha\}_{\alpha\in[\ell]}$, $\{\mathsf{K}^0_{\ell+\alpha}, \mathsf{K}^1_{\ell+\alpha}\}_{\alpha\in[\ell]}$, $\{\mathsf{K}^0_{2\ell+\alpha}, \mathsf{K}^1_{2\ell+\alpha}\}_{\alpha\in[2\ell]}$ correspond to the encoding information for the input of P_1, P_2 and the shares of P_3 respectively (w.l.o.g).

  ◦ Compute permutation strings $p_1, p_2 \in_R \{0,1\}^\ell$ for the garblers' input wires and generate commitments to e using randomness from $G(s)$. For $b \in \{0,1\}$, $(c^b_\alpha, \mathrm{o}^b_\alpha) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathsf{e}^{p_1^\alpha \oplus b}_\alpha)$,

$(c^b_{\ell+\alpha}, \mathsf{o}^b_{\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathsf{e}^{p_2^\alpha \oplus b}_{\ell+\alpha})$ for $\alpha \in [\ell]$ and $(c^b_{2\ell+\alpha}, \mathsf{o}^b_{2\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathsf{e}^b_{2\ell+\alpha})$ for $\alpha \in [2\ell]$. Set $\mathcal{B}_i = \left\{\mathsf{C}, \{c^b_\alpha\}_{\alpha\in[4\ell], b\in\{0,1\}}\right\}$. Broadcast $\mathcal{B}_i$.

- P_1 computes $m_1 = x_1 \oplus p_1$ and sends to P_3: the openings of the commitments corresponding to (x_1, x_{31}) i.e $\{\mathsf{o}_\alpha^{m_1^\alpha}, \mathsf{o}_{2\ell+\alpha}^{x_{31}^\alpha}\}_{\alpha\in[\ell]}$ and m_1. Similarly, P_2 computes $m_2 = x_2 \oplus p_2$ and sends to P_3: the openings of the commitments corresponding to (x_2, x_{32}) i.e $\{\mathsf{o}_{\ell+\alpha}^{m_2^\alpha}, \mathsf{o}_{3\ell+\alpha}^{x_{32}^\alpha}\}_{\alpha\in[\ell]}$ and m_2.

– Every party sets TTP as follows. If exactly one $P_i (i \in [2])$ broadcasts $(\texttt{Conflict}, P_3)$ in Round 2, set $\mathsf{TTP} = P_{[2]\setminus i}$. If both raise conflict, set $\mathsf{TTP} = P_1$. If $\mathcal{B}_1 \neq \mathcal{B}_2$, set $\mathsf{TTP} = P_3$.

Round 3: If $\mathsf{TTP} = \emptyset$, P_3 does the following:

- Assign $\mathsf{X}_1^\alpha = \mathsf{Open}(\mathsf{pp}, c_\alpha^{m_1^\alpha}, \mathsf{o}_\alpha^{m_1^\alpha})$ and $\mathsf{X}_{31}^\alpha = \mathsf{Open}(\mathsf{pp}, c_{2\ell+\alpha}^{x_{31}^\alpha}, \mathsf{o}_{2\ell+\alpha}^{x_{31}^\alpha})$ for $\alpha \in [\ell]$. Broadcast $(\texttt{Conflict}, P_1)$ if Open results in $\perp$
- Assign $\mathsf{X}_2^\alpha = \mathsf{Open}(\mathsf{pp}, c_{\ell+\alpha}^{m_2^\alpha}, \mathsf{o}_{\ell+\alpha}^{m_2^\alpha})$ and $\mathsf{X}_{32}^\alpha = \mathsf{Open}(\mathsf{pp}, c_{3\ell+\alpha}^{x_{32}^\alpha}, \mathsf{o}_{3\ell+\alpha}^{x_{32}^\alpha})$ for $\alpha \in [\ell]$. Broadcast $(\texttt{Conflict}, P_2)$ if Open results in $\perp$
- Else, set $\mathsf{X} = \mathsf{X}_1|\mathsf{X}_2|\mathsf{X}_{31}|\mathsf{X}_{32}$, run $\mathsf{Y} \leftarrow \mathsf{Ev}(\mathsf{C}, \mathsf{X})$ and $y \leftarrow \mathsf{sDe}(\mathsf{Y})$. Broadcast Y.

If P_3 broadcasts $(\texttt{Conflict}, P_i)$, then set $\mathsf{TTP} = P_{[2]\setminus i}$. If $\mathsf{TTP} = \emptyset$ and P_3 broadcasts Y, P_i $(i \in [2])$ does the following: Execute $y \leftarrow \mathsf{De}(\mathsf{Y}, \mathsf{d})$. If $y = \perp$, set $\mathsf{TTP} = P_1$.

Round 4: If $\mathsf{TTP} \neq \emptyset$: P_i $(i \in [2])$ sends x_i and o_{3i} (if valid) to TTP. P_3 sends $\mathsf{o}_{31}, \mathsf{o}_{32}$ to TTP.

Round 5: TTP computes $x_{3i} = \mathsf{Open}(c_{3i}, \mathsf{o}_{3i})$ using openings sent by P_1, P_2 (if available), else uses the openings sent by P_3. If valid opening is not received, a default value is used for shares of x_3. Compute $y = f(x_1, x_2, x_{31} \oplus x_{32})$ and send y to others.
Every party computes output as follows. If $y = \perp$ and received y' from TTP, set $y = y'$.

Figure 4.5: Protocol g3PC

4.6.1 Correctness and security

Below we give the proof of correctness.

Lemma 4.13 *A pair of honest parties can never be in conflict.*

Proof: It is easy to note that a pair of honest garblers will never be in conflict since the message $\mathcal{B}$ broadcast by them in Round 2 must be identical. Next, a garbler, say P_1 and evaluator P_3 would be in conflict only if one of the following hold: (a) The commitment and

opening of the input share sent by P_3 to P_1 is inconsistent (b) P_1's opening of committed encoded input of garbled circuit sent to P_3 fails. It is easy to check that the above cannot occur for honest P_1, P_3. □

Lemma 4.14 *An honest evaluator either evaluates the GC successfully at the end of round 2 or a* TTP *is established latest by Round 3.*

Proof: Consider an honest P_3. If a garbler raises a conflict with P_3 in Round 2, then his co-garbler is established as the TTP. Else, if P_3 receives broadcast and pairwise messages as per the protocol in round 2, then P_3 evaluates the circuit. On the other hand, if P_3 discovers that the broadcast messages sent by the garblers do not match, then P_3 is unanimously established as the TTP. Finally, in case P_3 locally identifies one of the garblers to be corrupt due to inconsistent/invalid pairwise message received in round 2, he raises a conflict, establishing the other garbler as the TTP. Thus the lemma holds. □

Theorem 4.6 *The protocol* g3PC *is correct i.e output obtained by the parties corresponds to a valid computation performed on unique set of inputs.*

Proof: We analyze the cases based on whether TTP is established during the protocol or not. If not, since none of the garblers raised a conflict with P_3 in Round 2, each of them must have a valid opening corresponding to P_3's public commitment of its input shares. In such a case, these shares constitute P_3's committed input. With respect to garblers, input labels sent by them in round 2 corresponding to their own input establish their committed inputs. It now follows from correctness of garbling and authenticity (potentially corrupt P_3 could not have forged Y) that the output obtained by all corresponds to the evaluation of garbled circuit on above mentioned committed inputs. We now consider the case when TTP is established. Here, the inputs sent by garblers on clear to the TTP constitute their committed inputs. The committed input of P_3 depends on whether the TTP is established during or after Round 2. In the former where none of the garblers raised conflict in Round 2, it is clear from the protocol description that P_3's committed input is based on its shares distributed in Round 1 (enforced by binding of commitment on input shares). Else, the committed input of P_3 is considered as the one sent on clear to the TTP. Finally, the correctness of output computation based on committed inputs follows from the fact that the TTP must be honest (Lemma 4.13 shows that the pair of parties in conflict must involve the corrupt). □

While the full proof of security appear in Section 4.8.4, the intuition on why the protocol achieves god and the theorem statement follow. Based on whether the evaluator is honest or

corrupt, god is argued below. By Lemma 4.14, an honest evaluator either identifies a TTP or evaluates the GC successfully at the end of round 2. If evaluation is performed, then an honest evaluator would obtain output by soft decoding and enable the garblers to get output by sending the encoded output. If TTP is identified by an honest evaluator all parties accept the output sent by the TTP. Next, consider a corrupt evaluator. In case a corrupt evaluator does not communicate the encoded output to the garblers or sends an invalid Y, then the garblers would unanimously identify the evaluator to be corrupt. Then, P_1 would be chosen as a TTP and eventually each party receives the output through the computation performed by TTP. Even in the case when a corrupt evaluator falsely raises a conflict, the TTP chosen by him must be honest and each party would obtain the output from the TTP.

Theorem 4.7 *Assuming one-way functions, protocol* g3PC *securely realizes the functionality* $\mathcal{F}_{\mathsf{god}}$ *(Figure 2.4) against a malicious adversary that corrupts at most one party.*

4.6.2 Optimizations

We propose several optimizations for g3PC to reduce its communication. Firstly, since broadcast communication is considered more expensive than private communication, a broadcast of a message, say m is replaced with broadcast of $\mathsf{H}'(m)$, where H' denotes a collision-resistant hash while the message m is sent privately over point-to-point channel to the receiver. Besides, the trick described for f3PC (Section 4.3.2) can be applied where the common message of garblers $\mathcal{B}$ is divided into equal halves $\mathcal{B} = \mathcal{B}^1||\mathcal{B}^2$; each garbler sends one part on clear and the other in compressed form. Second, we elaborate on the optimization applied to broadcast of Y in round 3 by P_3: P_3 broadcasts $\mathsf{H}'(\mathsf{Y})$ where Y denotes the encoded output comprising of concatenation of the output label of each output wire obtained by GC evaluation. Additionally, P_3 sends Y privately to each of the garblers enabling them to compute the hash of the message received privately and check against the broadcast message to conclude its consistency. Thus, the optimization applied on broadcast of $\mathcal{B}$ and Y makes broadcast independent of circuit size. Finally, we point that the description of protocol in Figure 4.5 includes certain redundancies such as a party established as TTP sending message to itself and the protocol proceeding till the last round even in cases where termination can occur earlier. This was done only to keep the protocol description simple and facilitate better understanding. In the implementation, the redundant messages are avoided. Further, when TTP is established in round 2 itself, round 3 can be skipped and the last two rounds executed, enabling the protocol to terminate within 4 rounds.

Table 4.3: Computation time (**CT**), Runtime for LAN (**LAN**), WAN (**WAN**) and Communication (**CC**) for the 3PC of [159].

Circuit	CT(ms)		LAN(ms)		WAN(s)		CC(KB)	
	P_1/P_2	P_3	P_1/P_2	P_3	P_1/P_2	P_3 (s)	P_1/P_2	P_3
AES	0.96	0.72	1.19	0.86	0.62	1.04	153.2	2.1
SHA-256	11.36	9.4	13.3	10.7	1.05	1.65	3073.6	4.5
MD5	4.5	3.0	4.9	3.9	0.83	1.24	1036.4	2.5

4.7 Experimental results

In this section, we provide empirical results for our protocols. We use the circuits of AES-128, SHA-256 and MD5 as benchmarks. We start with the description of the setup environment, both software and hardware.

Hardware Details. We have experimented both in LAN and WAN setting. The specifications of our systems used for LAN include 32GB RAM; an Intel Core i7-7700-4690 octa-core CPU with 3.6 GHz processing speed. The hardware supports AES-NI instructions. For WAN setting, we use Microsoft Azure Cloud Services with machines located in West USA, East Asia and India. Our 3PC protocols have exactly one party at each location while for 4PC results, two of the four parties are located in East Asia and one party each in West USA and India. We used machines with 1.75GB RAM and single core processor. The bandwidth is limited to 100Mbps for the WAN network between the machines in West USA and East Asia and it is limited to 8Mbps from the machine in India. Before running our experiments, we measured sample round trip delay between India-West USA, India-East Asia and East Asia-West USA for communication of one byte of data. These values average to 0.42 s, 0.14 s and 0.18 s respectively.

Software Details. For efficient implementation, the garbling technique used throughout is that of Half Gates [183]. The code is built on libgarble library whose starting point is the JustGarble library, both licensed under GNU GPL License. The libgarble library operates with AES-NI support from hardware. The operating system used for LAN and WAN results are Ubuntu 17.10 (64-bit) and Ubuntu 16.04 (64-bit) respectively. Our code follows the standards of C++11. We make use of openSSL 1.0.2g library for commitments. We use SHA-256 to implement a commitment scheme. We have benchmarked our results with 3 circuits AES, SHA-256, MD5. The circuit description is obtained as a simple text file (.txt) for implementation purposes. Communication is done with the help of sockets. We instantiate multiple threads to facilitate communication between the garblers and evaluator. The garblers also share a connection between each other to share the randomness. All our results indicate the average values over a set of 20 runs of the experiments.

Table 4.4: Computation time (**CT**), Runtime for LAN (**LAN**), WAN (**WAN**) and Communication (**CC**) for f3PC protocol.

Circuit	CT(ms)		LAN (ms)		WAN (s)		CC(KB)	
	P_1/P_2	P_3	P_1/P_2	P_3	P_1/P_2	P_3	P_1/P_2	P_3
AES	1.04	0.74	1.17	1.0	0.83	1.27	161.55	2.27
SHA-256	11.55	9.5	13.6	12.5	1.65	1.97	3089.7	4.5
MD5	4.61	3.05	4.96	4.32	1.39	1.54	1044.93	2.52

Table 4.5: Computation time (**CT**), Runtime for LAN (**LAN**), WAN (**WAN**) and Communication (**CC**) for g4PC protocol.

Circuit	CT(ms)			LAN(ms)			WAN(s)			CC(KB)		
	P_1/P_2	P_3	P_4	P_1/P_2	P_3	P_4	P_1/P_2	P_3	P_4	P_1/P_2	P_3	P_4
AES	0.95	0.8	0.04	1.21	0.96	0.27	0.78	1.08	0.47	163.3	8.1	2.1
SHA-256	11.3	9.72	0.09	13.67	12.06	0.54	1.86	2.0	0.54	3091.9	14.1	2.1
MD5	4.42	3.03	0.07	5.05	4.1	0.43	1.24	1.66	0.52	1046.8	8.13	2.1

Table 4.6: Computation time (**CT**), Runtime for LAN (**LAN**) and Communication (**CC**) both over private (**pp**) and broadcast (**bc**) channels for g3PC protocol.

Circuit	CT(ms)		LAN(ms)		pp CC(KB)		bc CC(KB)	
	P_1/P_2	P_3	P_1/P_2	P_3	P_1/P_2	P_3	P_1/P_2	P_3
AES	1.12	0.9	2.62	2.58	153.36	2.23	0.032	0.06
SHA-256	11.63	9.76	16.25	13.8	3074.16	4.62	0.032	0.06
MD5	4.73	3.22	7.18	5.88	1036.66	2.51	0.032	0.06

Comparison. We compare our results with the related ones for the high-latency networks (such as the Internet) in the honest majority setting. The most relevant is that of [159] and we elaborate on the comparison with it below. With regard to the 4-party protocol of [129], it is expected to lag in performance compared to g4PC since its computation and communication is significantly higher. As per our calculations, the overhead of transmitting 12 GCs instead of 1 is more than the efficiency gain of having 2 rounds instead of 5, even with bandwidth of 100Mbps for our benchmark circuits of SHA-256 and MD5. In case of limited bandwidth of around 8Mbps, our protocol would perform better than that of [129] for all our benchmark circuits including AES. The difference in performance will be even more significant for larger circuits or when multiple MPC executions are run in parallel. Another work close to our setting is that of [52] that explores 5PC in the honest majority setting. Similar to [159], it only provides sa. It uses distributed garbling and requires 8 rounds. Our 3 party and 4 party protocols perform better than the protocol of [52], in spite of achieving better security notions of fn and god. The total communication for any of our protocol constitutes only 1 - 3.5 % of the total communication of their implementation in the malicious setting and 3 - 6 % of the total communication of their implementation in the semi-honest setting.

For comparing with [159], four parameters are considered– computation time (**CT**), communication cost (**CC**) and runtime both in LAN (**LAN**) and WAN (**WAN**). The LAN and WAN

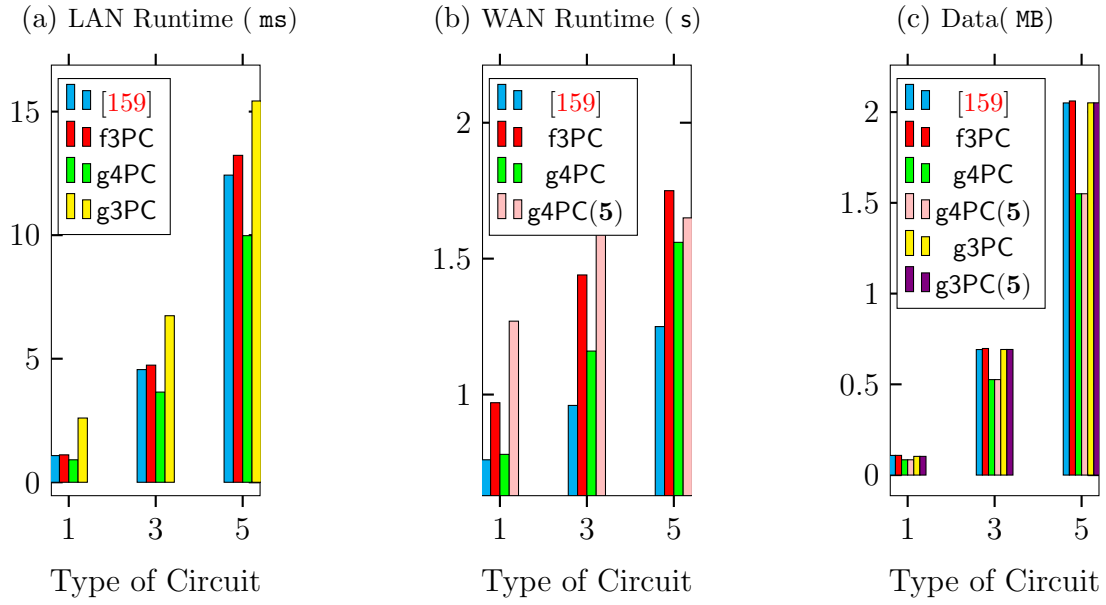
Table 4.7: The average computation time (**aCT**), runtime in LAN (**aLAN**), WAN (**aWAN**) and communication (**aCC**) per party for [159] and our protocols. The figures in bracket indicate the increase for the worst case 5-round runs of g4PC and g3PC.

Circuit	aCT(ms)				aLAN(ms)				aWAN(s)			aCC(KB)			
	[159]	f3PC	g4PC	g3PC	[159]	f3PC	g4PC	g3PC	[159]	f3PC	g4PC	[159]	f3PC	g4PC	g3PC
AES	0.88	0.94	0.69	1.04	1.08	1.11	0.91	2.60	0.76	0.97	0.78 (+.49)	102.83	108.46	84.2 (+.01)	103.02 (+.02)
SHA-256	10.70	10.87	8.1	11.01	12.43	13.23	9.98	15.43	1.25	1.75	1.56 (+.52)	2050.56	2061.3	1550 (+.1)	2051.02 (+.08)
MD5	4.0	4.09	2.98	4.22	4.56	4.74	3.65	6.74	0.96	1.44	1.16 (+.49)	691.76	697.46	525.97 (+.03)	691.98 (+.09)

runtime are computed by adding the computation time and the corresponding network time. Noting that the roles of the parties in the protocols are asymmetric, we show the computation time, LAN and WAN runtime and communication cost separately for the parties with distinct roles. The trend of WAN runtime across the tables indicates the influence of round complexity and the location of servers. For a fair comparison with our protocols, we instantiate the protocol of [159] in our environment and the results appear in Table 4.3. The results for our 3PC with fn, 4PC (5 rounds) and 3PC with god appear in Tables 4.4, 4.5 and 4.6 respectively. With respect to our 4-round 4PC with god, in the worst case run, we save a round at the expense of one garbled circuit over our 5-round 4PC which amounts to 72 KB − 1530 KB for the benchmark circuits. For the 3PC with god, we provide implementation result only for LAN setting where the broadcast channel is emulated using an UDP physical broadcast. We calculate separately the cost of communication over private channels and broadcast channel and demonstrate that the latter communication is independent of the circuit size. Our protocols providing god run in 3 rounds when the adversary does *not* strike. The round complexity stretches to 5 in the worst case for our 5-round protocols. Tables 4.5-4.6 show performance for the 3-round runs. With minimal communication and computation in the last two rounds, the overhead shows up mainly in the WAN runtime by a factor of half a second and communication by less than 1 KB.

For a unified comparison with [159], we compute the average of the above parameters per party for all the protocols and the results appear in Table 4.7. In terms of average computation time, LAN runtime and communication cost our 4PC turns out to be the winner inspite of providing the strongest notion of security. The improvement per party comes from the fact that the costs of this protocol are almost similar to that of 3PC protocols inspite of having one extra party in the system. It closely trails [159] in terms of WAN runtime due to the additional communication involved in the InputCommit routine and the delayed opening of the committed decoding information both of which are not present in the protocol of [159]. Our 3PC with fn is almost on par with [159] and yet achieves a stronger security notion. The extra overhead over [159] occurs primarily as a consequence of commitments to the decoding information and the postponed opening of decoding information by the garblers in order to achieve fn. However,

Figure 4.6: Performance Comparison (avg/party) of various Protocols for fn and god. (**5**) denotes worst case execution of the protocol in consideration.



The x-axes indicate the type of the circuit used for evaluation 1-AES, 3-MD5, 5-SHA-256. The y-axis indicates Runtime in ms, s for graphs (a), (b) respectively and communication data in MB for (c).

in [159], the use of soft-decoding avoided the need for additional communication to deliver the decoding information. The variation in the communication overhead over the circuits reflects the fact that the output size and thus the size of information (openings of the commitments) related to decoding information are different over the circuits. For example, the SHA-256 has 256 bit output, whereas the output size of AES is half of it. Therefore, the communication overhead for SHA-256 for our protocol is almost double that of AES, namely 10.74 KB vs. 5.63 KB. The WAN runtime overhead reflects the increased round requirement of our fair protocol. The communication overhead of our 3PC with god is almost nominal over [159] as both protocols use of soft-decoding. In Table 4.7, we show in bracket the increase for the 5-round runs of our 4PC (5-round) and 3PC protocols providing god. The performance of our protocols compared to that of [159] is plotted in Figure. 4.6.

4.8 Security Proofs

4.8.1 Security Proof of f3PC Protocol

In this section, we provide a complete proof for the Theorem 4.2 that states the security of f3PC relative to its ideal functionality.

We first explain the technicality behind using an equivocal commitment scheme (eNICOM) to commit to the decoding information. In our protocol, the adversary can decide whether to let the computation succeed or fail till round 3. This forces the simulator to make the same decision on adversary's behalf at the end of round 3. As a result, the simulator can get access to the output, only after simulation of round three is completed, at the earliest. Therefore, the simulator needs to send the GC, encoding information and the commitment on decoding information without access to the output, while acting on behalf of the honest parties. This is achieved by invoking oblivious simulator of GC which neither takes the output, nor returns the decoding information. Consequently, the simulator commits to a dummy value in round 2. Later if and when $\mathcal{F}_{\mathsf{fair}}$ is invoked and y is known, $\mathcal{S}_{\mathsf{prv}}$ is invoked with the same randomness which simply returns the decoding information that makes the fake GC returned by $\mathcal{S}_{\mathsf{obv}}$ output y. Correspondingly, the simulator equivocates to the correct decoding information that it obtains from the privacy simulator in round 4. Equivocality is enabled via a trapdoor which in our protocol remains distributed between the garblers. The public parameter for eNICOM is generated jointly by the garblers (Section 2.4.2.1).

We now describe the simulator $\mathcal{S}_{\mathsf{f3PC}}$ for the case when P_1, P_3 is corrupt. The case of P_2 being corrupt is symmetric to that of P_1. Since the protocol may result either in output computation or abort based on the corrupt party's behaviour until Round 3, the privacy simulator $\mathcal{S}_{\mathsf{prv}}$ (Ref. [27]) that demands the output can only be invoked only at the end of Round 3. Therefore, the oblivious simulator of the garbling scheme $\mathcal{S}_{\mathsf{obv}}$ (Ref. [27]) that does not need output is invoked first as a part of GC generation. We assume a garbling scheme such that $\mathcal{S}_{\mathsf{obv}}$ and $\mathcal{S}_{\mathsf{prv}}$ when invoked on same randomness return the same (C, X) (Most known garbling schemes based on Yao comply with this [182, 183, 141]). Later, if the adversary behaves such that the protocol results in output computation, the evaluator's input is extracted, used to obtain output y via $\mathcal{F}_{\mathsf{fair}}$ and $\mathcal{S}_{\mathsf{prv}}$ is invoked to retrieve decoding information. Since this can be done earliest after Round 3, we use an equivocal commitment to explain the commitment on decoding information sent in Round 2. The description of simulator $\mathcal{S}^3_{\mathsf{f3PC}}$ corresponding to P_3 (evaluator) corrupt and $\mathcal{S}^1_{\mathsf{f3PC}}$ corresponding to P_1 (garbler) corrupt is available in Figure 4.7 with **R1/R2/R3/R4**

indicating simulation for round 1, 2, 3 and 4 respectively.

Simulator $\mathcal{S}_{\mathsf{f3PC}}$

$\underline{\mathcal{S}^3_{\mathsf{f3PC}}\ (P_3^*\ \textbf{is corrupt})}$

R1 Receive (pp^1, x_{31}) and (pp^2, x_{32}) privately from P_3^* on the behalf of P_1, P_2 respectively. If the input share is not received / invalid, consider a default value.

R1 Send (h_1, r_1) and (h_2, r_2) to P_3 according to the protocol on behalf of P_1, P_2 respectively.

R2 Use uniform randomness r on behalf of P_1, P_2 and run $(\mathsf{C}, \mathsf{X}) \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C)$, where $\mathcal{S}_{\mathsf{obv}}$ is the oblivious simulator of the garbling scheme.

R2 Choose m_1, m_2 at random. Let $\{c_\alpha^{m_1^\alpha}, c_{\ell+\alpha}^{m_2^\alpha}, c_{2\ell+\alpha}^{x_{31}^\alpha}, c_{3\ell+\alpha}^{x_{32}^\alpha}\}_{\alpha\in[\ell]}$ be commitments to the entries of X, corresponding to pp^1. If $\mathsf{pp}^1 \neq \mathsf{pp}^2$, the above is computed with respect to pp^2 as well. Commit to dummy values corresponding to other input wire labels. Using eCom (sample epp with trapdoor t_1, t_2), create c as a commitment to a dummy value (Incase of Naor-based NICOM, set c to the specific commitment supporting equivocation). Set $\mathcal{B}_i$ $(i \in [2])$ to include C, the set of commitments computed with respect to pp^i and c. Send $\mathcal{B}_i$ on behalf of P_i. Send $(\{\mathrm{o}_\alpha^{m_1^\alpha}, \mathrm{o}_{2\ell+\alpha}^{x_{31}^\alpha}\}_{\alpha\in[\ell]}, m_1)$, $(\{\mathrm{o}_{\ell+\alpha}^{m_2^\alpha}, \mathrm{o}_{3\ell+\alpha}^{x_{32}^\alpha}\}_{\alpha\in[\ell]}, m_2)$ on behalf of P_1, P_2 to P_3^*.

R4 Suppose on behalf of some P_i $(i \in [2], j \in [2] \setminus i)$ received $(\mathsf{Y} = \mathsf{Ev}(\mathsf{C}, \mathsf{X}), r'_j)$ from P_3^* in Round 3 such that $\mathsf{H}(r'_j) = h_j$. Then invoke $\mathcal{F}_{\mathsf{fair}}$ with (Input, x_3) on behalf of P_3^* (where x_3 is computed as $x_3 = x_{31} \oplus x_{32}$) to obtain output y. Run $(\mathsf{C}, \mathsf{X}, \mathsf{d}') \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y)$ where $\mathcal{S}_{\mathsf{prv}}$ refers to the privacy simulator of the garbling scheme. Send o to P_3^* on behalf of P_i where $\mathrm{o} = \mathsf{Equiv}(c, \mathsf{d}', t_1, t_2)$.

R4 Else invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \mathtt{abort})$ on behalf of P_3^*.

$\underline{\mathcal{S}^1_{\mathsf{f3PC}}\ (P_1^*\ \textbf{is corrupt})}$

R1 Send a random share x_{31} and pp on behalf of P_3. Choose r_2 uniformly at random to compute $h_2 = \mathsf{H}(r_2)$. Send (epp_2, h_2) to P_1^* on behalf of P_2 according to the protocol.

R1 Receive (s, h_1, epp_1) on behalf of P_2 and (h_1, r_1) on behalf of P_3. Compute $\mathcal{B}$ on behalf of P_2 as per protocol.

R2 Invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{sid}, \mathsf{Input}, \mathtt{abort})$ on behalf of P_1^* and set $y = \perp$ if (a) h_1 received on behalf of P_2, P_3 does not match or $\mathsf{H}(r_1) \neq h_1$ or (b) $\mathcal{B}$ received from P_1^* on behalf of P_3 does not match the $\mathcal{B}$ computed on behalf of P_2 or (c) any of the decommitments corresponding to encoded

inputs sent by P_1^* to P_3 opens to something other than what was originally committed (known on behalf of P_2).

R2 Else, extract P_1^*'s input as $x_1 = m_1 \oplus p_1$, where p_1, m_1 is known on behalf of P_2, P_3 respectively. Invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{sid}, \mathsf{Input}, x_1)$ to get output y.

R3 Compute Y such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$ (d known on behalf of P_2). Send (Y, r_2) to P_1^* on behalf of P_3.

R4 If $y \neq \perp$, send (y, r_1) to P_1^* on behalf of P_2.

Figure 4.7: Description of $\mathcal{S}_{\mathsf{f3PC}}$

Security against corrupt P_3^*. We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}^3_{\mathsf{f3PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{f3PC}, \mathcal{A}}$, when $\mathcal{A}$ corrupts P_3. The views are shown to be indistinguishable via a series of intermediate hybrids.

– HYB_0: Same as $\text{REAL}_{\mathsf{f3PC}, \mathcal{A}}$.

– HYB_1: Same as HYB_0, except that P_1, P_2 use uniform randomness rather than pseudorandomness.

– HYB_2: Same as HYB_1, except that some of the commitments of input wire labels sent by P_1, P_2, which will not be opened are replaced with commitments of dummy values. Specifically, these are the commitments with indices $\neq m_1, m_2, x_{31}, x_{32}$.

– HYB_3 : Same as HYB_2, except the following:

- $\text{HYB}_{3.1}$: When the execution results in `abort`, the GC is created as $(\mathsf{C}', \mathsf{X}) \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C)$ and the commitment to the decoding information is created for a dummy value.
- $\text{HYB}_{3.2}$: When the execution results in output y, the GC is created as $(\mathsf{C}', \mathsf{X}, d') \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y)$, the commitment c to the decoding information is created for a dummy value and later equivocated to d' using o computed via $\mathsf{o} \leftarrow \mathsf{Equiv}(c, \mathsf{d}', t_1, t_2)$.

– HYB_4: Same as HYB_3, except that the protocol results in abort if neither P_1 nor P_2 receive Y obtained upon GC evaluation from P_3.

Since $\text{HYB}_4 := \text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}_{\mathsf{f3PC}}}$, we show that every two consecutive hybrids are computationally indistinguishable which concludes the proof.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that P_1, P_2 use uniform randomness in HYB_1 rather than pseudorandomness as in HYB_0. The indistinguishability follows via reduction

to the security of the PRG G.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is that some of commitments of the input labels in HYB_1 that will not be opened are replaced with commitments of dummy values in HYB_2. The indistinguishability follows via reduction to the hiding property of Com that holds even though pp was chosen by corrupt P_3.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_{3.1}$: The difference between the hybrids is in the way (C, X) is generated when the execution results in $\mathtt{abort}$. In HYB_2, $(\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C')$ is run, which gives $(\mathsf{C}, \mathsf{En}(x, \mathsf{e}))$. In $\text{HYB}_{3.1}$, it is generated as $(\mathsf{C}', \mathsf{X}) \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C')$. Additionally, the commitment to the decoding information is created for a dummy value in $\text{HYB}_{3.1}$. The indistinguishability follows via reduction to the obliviousness of garbling and the hiding property of eCom.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_{3.2}$: The difference between the hybrids is in the way $(\mathsf{C}, \mathsf{X}, \mathsf{d})$ is generated. In HYB_2, $(\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C')$ is run, which gives $(\mathsf{C}, \mathsf{En}(x, \mathsf{e}), \mathsf{d})$. In $\text{HYB}_{3.2}$, it is generated as $(\mathsf{C}', \mathsf{X}, \mathsf{d}') \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C', y)$. Additionally, the commitment to the decoding information is created for a dummy value and later equivocated to d' using o computed via $\mathsf{o} \leftarrow \mathsf{Equiv}(c, \mathsf{d}', t_1, t_2)$. The indistinguishability follows via reduction to the privacy of the garbling scheme and the hiding property of eCom.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The difference between the hybrids is that in HYB_3, the protocol results in abort if neither P_1 nor P_2 receive Y such that $\mathsf{De}(\mathsf{Y}, d) \neq \perp$ from P_3; while in HYB_4, the protocol results in abort if neither P_1 nor P_2 receive the Y that P_3 obtained upon GC evaluation. Due to authenticity of the garbling scheme, P_3 could have sent Y such that $\mathsf{Y} \neq \mathsf{Ev}(\mathsf{C}, \mathsf{X})$ but $\mathsf{De}(\mathsf{Y}, d) \neq \perp$ only with negligibility probability. Therefore, the hybrids are indistiguishable.

Security against corrupt P_1^*. We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}^1_{\mathsf{f3PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{f3PC}, \mathcal{A}}$, when $\mathcal{A}$ corrupts P_1. The views are shown to be indistinguishable via a series of intermediate hybrids.

- HYB_0: Same as $\text{REAL}_{\mathsf{f3PC}, \mathcal{A}}$.
- HYB_1: Same as HYB_0, except that P_3 aborts if it accepts any decommitment that opens to a value other than what was originally committed.
- HYB_2: Same as HYB_1, except that Y is computed via $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$ rather that $\mathsf{Y} = \mathsf{Ev}(\mathsf{C}, \mathsf{X})$.

– HYB_3: Same as HYB_2, except that P_2 outputs $\perp$ if GC could not be evaluated by P_3 successfully.

Since $\text{HYB}_3 := \text{IDEAL}_{\mathcal{F}_{\mathsf{fair}},\mathcal{S}_{\mathsf{Fair}}}$, we show that every two consecutive hybrids are computationally indistinguishable which concludes the proof.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that in HYB_0, P_3 aborts if the decommitments sent by P_1 output $\perp$ while in HYB_1, P_3 aborts if the decommitments sent by P_1 opens to any value other than what was originally committed. Since the commitment scheme Com is binding and pp was chosen uniformly at random by P_3, in HYB_0, P_1 could have decommitted successfully to a different input label than what was originally committed, only with negligible probability.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is that in HYB_1, P_3 computes Y via $\mathsf{Ev}(\mathsf{C},\mathsf{X})$, while in HYB_2, Y is computed such that $\mathsf{De}(\mathsf{Y},\mathsf{d}) = y$. Due to the correctness of the garbling scheme, the equivalence of Y computed via $\mathsf{Ev}(\mathsf{C},\mathsf{X})$ or such that $\mathsf{De}(\mathsf{Y},\mathsf{d}) = y$ holds.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is that in HYB_2, P_2 may output non-$\perp$ if it receives a valid 'proof' from P_1 even though P_3 was unable to evaluate the GC successfully, while in HYB_3, P_2 outputs $\perp$ in this scenario. Due to the preimage resistance property of Hash H, P_1 could have been able to compute a valid proof i.e r_2' such that $\mathsf{H}(r_2') = h_2$ only with negligible probability.

4.8.2 Security Proof for g4PC

In this section, we present the complete security proof of the Theorem. 4.3 that states the security of $\mathsf{g4PC}$ relative to its ideal functionality.

We describe the simulator $\mathcal{S}_{\mathsf{g4PC}}$ for the case when P_1, P_3 and P_4 is corrupt. The simulator acts on behalf of all the honest parties in the execution. The corruption of P_2 is symmetric to the case when P_1 is corrupt. For better clarity, we separate out the simulation for the subroutine $\mathsf{InputCommit}_i$. Specifically, we describe the simulator corresponding to $\mathsf{InputCommit}_1$ (simulation of $\mathsf{InputCommit}_2, \mathsf{InputCommit}_3, \mathsf{InputCommit}_4$ follow analogously) for the case of corrupt P_1 and P_2. The cases of P_3, P_4 being corrupt during $\mathsf{InputCommit}_1$ is symmetric to the case of P_2. Figure 4.8 and Figure 4.9 describes the simulator with **R1, R2, R3, R4, R5** depicting simulation for rounds 1, 2, 3, 4 and 5 respectively.

We first give brief overview of the main technicalities of the simulator. During simulation

of $\mathsf{InputCommit}_i$ corresponding to corrupt P_i, it is possible for the simulator acting on behalf of the honest parties to extract the committed input of the corrupt in the first round itself based on whether P_i had sent consistent messages to at least majority of the honest parties (else a default value is used). Thus, the extracted input can be used to obtain output y via $\mathcal{F}_{\mathsf{god}}$ at the end of Round 1 of simulation. The main technicality arises with respect to simulation in case of corrupt P_3. In this case, either the oblivious simulator of the garbling scheme $\mathcal{S}_{\mathsf{obv}}$ (Ref. [27]) or the privacy simulator $\mathcal{S}_{\mathsf{prv}}$ (can be invoked with output y obtained) is invoked based on whether corrupt P_3 would get access to input labels corresponding to any of his non-committed input shares or not respectively in Round 2. This is known by the simulator acting on behalf of both the honest garblers since the committed input of the corrupt P_3 is known to simulator at end of Round 1. Finally in the former case when GC returned by $\mathcal{S}_{\mathsf{obv}}$ is used, the commitment on hash of decoding information is dummy (never has to be opened); while in the latter case when GC returned by $\mathcal{S}_{\mathsf{prv}}$ is used, commitment on hash of decoding information is done on the value d returned by the simulator. With this background, we now proceed to the formal description.

Simulator $\mathcal{S}_{\mathsf{InputCommit}_1}$

$\mathcal{S}^1_{\mathsf{InputCommit}_1}$ **(P_1^* is corrupt)**

R1 Receive commitments $\mathsf{c}_{12}, \mathsf{c}_{13}, \mathsf{c}_{14}$ on behalf of each among P_2, P_3, P_4. Receive o_{12} on behalf of P_3, P_4; o_{13} on behalf of P_2, P_4 and o_{14} on behalf of P_2, P_3.

R1 Set $\mathcal{C}_k = P_1$ on behalf of P_k ($k \in \{2,3,4\}$) if $\mathsf{sOpen}(\mathsf{pp}_1, \mathsf{c}_{1j}, \mathsf{o}_{1j})$ ($j \in \mathsf{ind}(\mathcal{P}_{1k})$) received from P_1^* results in $\perp$.

R1 If there does not exist majority in the versions of $(\mathsf{pp}_1, \mathsf{c}_{12}, \mathsf{c}_{13}, \mathsf{c}_{14})$ received on behalf of P_2, P_3, P_4 from P_1^*, assume a default value for P_1's input share and add P_1^* to $\mathcal{C}_k$, where $k \in \{2,3,4\}$.

R1 Else, set $(\mathsf{pp}_1, \mathsf{c}_{12}, \mathsf{c}_{13}, \mathsf{c}_{14})$ as the majority value and $(\mathsf{o}_{12}, \mathsf{o}_{13}, \mathsf{o}_{14})$ as the corresponding opening. Compute $x_1 = x_{12} \oplus x_{13} \oplus x_{14}$ where $x_{1j} = \mathsf{sOpen}(\mathsf{pp}_1, \mathsf{c}_{1j}, \mathsf{o}_{1j})$ for $j \in \{2,3,4\}$. Invoke $\mathcal{F}_{\mathsf{god}}$ with (Input, x_1) on behalf of P_1^* to obtain output y.

R1 If received different versions of $(\mathsf{pp}_1, \mathsf{c}_{12}, \mathsf{c}_{13}, \mathsf{c}_{14})$ on behalf of P_α, P_β (where $\alpha, \beta \in \{2,3,4\}$), add (P_1, P_α) in $\mathcal{F}_\beta$ and (P_1, P_β) in $\mathcal{F}_\alpha$.

$\mathcal{S}^2_{\mathsf{InputCommit}_1}$ **(P_2^* is corrupt)**

R1 On behalf of P_1: Sample pp_1 and compute c_{1j} as commitments on randomly chosen x_{1j} for $j \in \mathsf{ind}(\mathcal{P}_{12})$ (input shares of P_1 available to corrupt P_2^*) and commitment of dummy value corresponding to $j \notin \mathsf{ind}(\mathcal{P}_{12})$. Send $(\mathsf{pp}_1, \mathsf{c}_{12}, \mathsf{c}_{13}, \mathsf{c}_{14})$ and openings $(\mathsf{o}_{13}, \mathsf{o}_{14})$ to P_2^*.

R2 Send $(\mathsf{pp}_1, \mathsf{c}_{12}, \mathsf{c}_{13}, \mathsf{c}_{14})$ and o_{14} to P_2^* on behalf of P_3. Send $(\mathsf{pp}_1, \mathsf{c}_{12}, \mathsf{c}_{13}, \mathsf{c}_{14})$ and o_{13} to P_2^* on behalf of P_4 .

R2 Receive $(\mathsf{pp}'_1, \mathsf{c}'_{12}, \mathsf{c}'_{13}, \mathsf{c}'_{14})$ from P_2^* on behalf of P_k ($k \in \{3,4\}$). Add (P_1, P_2) to $\mathcal{F}_k$ if the version received from P_2^* is not identical to the one sent on behalf of P_1 in Round 1. Additionally, receive o'_{13}, o'_{14} on behalf of P_4 and P_3 respectively. Add P_2 to $\mathcal{C}_k$ ($k \in \{3,4\}$) if the opening received on behalf of P_k is anything other than what was originally sent on behalf of P_1 in Round 1.

Figure 4.8: Description of $\mathcal{S}_{\mathsf{InputCommit}_1}$

Simulator $\mathcal{S}_{\mathsf{g4PC}}$

$\mathcal{S}^3_{\mathsf{g4PC}}$ **(P_3^* is corrupt)**

R1 Simulation of Round 1 of $\mathcal{S}^3_{\mathsf{InputCommit}_\alpha}$ ($\alpha \in [4]$) (Figure 4.8). Let y denote the output computed.

R1 Receive pp_3^1 and pp_3^2 from P_3^* on behalf of P_1 and P_2 respectively.

R2 Simulation of Round 2 of $\mathcal{S}^3_{\mathsf{InputCommit}_\alpha}$ ($\alpha \in [4]$) (Figure 4.8).

R2 If $P_3 \in \mathcal{C}_i$ ($i \in \{1,2\}$) or $(P_1, P_3) \in \mathcal{F}_2$ or $(P_2, P_3) \in \mathcal{F}_1$ (i.e an honest garbler may not have access to P_3's committed share at end of Round 1), use uniform randomness r on behalf of P_1, P_2 instead of pseudorandomness and run $(\mathsf{C}', \mathsf{X}') \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C)$, where $\mathcal{S}_{\mathsf{obv}}$ is the oblivious simulator of the garbling scheme. Choose $\{m_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_i)}$ at random. Let $m_i \leftarrow ||_{j\in\mathsf{ind}(\mathcal{P}_i)} m_{ij}$ and $\{c_\alpha^{m_1^\alpha}, c_{3\ell+\alpha}^{m_2^\alpha}, c_{6\ell+\alpha}^{m_3^\alpha}, c_{9\ell+\alpha}^{m_4^\alpha}\}_{\alpha\in[\ell]}$ be commitments to the entries of X, corresponding to pp_3^1. Commit to dummy values corresponding to other input wire labels. Let $\mathcal{B}^1 = \left\{\mathsf{C}', \{c_\alpha^b\}_{\alpha\in[12\ell], b\in\{0,1\}}, c', \{p_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{i3})}\right\}$ where p_{ij}'s are computed as follows: With respect to $i \in \mathsf{ind}(\mathcal{P}_3), j \in \mathsf{ind}(\mathcal{P}_{i3})$, it is computed as $p_{ij} = x_{ij} \oplus m_{ij}$ consistent with the (opening of) shares distributed to P_3^* during simulation of $\mathsf{InputCommit}_i$. Corresponding to P_3's shares, it is computed with respect to the opening received on behalf of P_1 (if valid, else take default) during simulation of $\mathsf{InputCommit}_3$. Here, c' is a commitment to dummy value. Send $\mathcal{B}^1$ to P_3^* on behalf of P_1. If $P_3 \notin \mathcal{C}_1$, additionally send M_1 and (openings of) encoding information corresponding to indices $\{m_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{i1})}$ (corresponding to $\{x_{13}, x_{14}, x_{34}\}$) as per protocol. Analogous steps are executed on behalf of P_2.

R2 Else, run $(\mathsf{C}', \mathsf{X}', \mathsf{d}') \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y)$. Execute similar steps as above except that c' is computed as commitment on $\mathsf{H}(\mathsf{d}')$.

R3 If $\mathcal{C}_\alpha \neq \emptyset \vee \mathcal{F}_\alpha \neq \emptyset$, P_α $(\alpha \in \{1,2,4\})$, send (TTP, β) to P_3^* where $P_\beta \notin \mathcal{C}_\alpha \cup \mathcal{F}_\alpha$.

R3 If $\mathcal{C}_g = \mathcal{F}_g = \emptyset$ $(g \in [2])$ send opening of hash of decoding information o to P_3^* on behalf of P_g.

R4 If received $\mathsf{Y} = \mathsf{Ev}(\mathsf{C}, \mathsf{X})$ from P_3 on behalf of $P_g (g \in [2])$, send y to P_3^* on behalf of P_g.

R4 If received a valid view V_3 from P_3^* (comprising of openings corresponding to P_3's committed shares and the shares sent on behalf of honest parties in Round 1) along with (TTP, l), $l \in [4] \setminus \{3\}$ on behalf of P_l during Round 3, send (y, TTP) to P_3^* in Round 4 on behalf of P_l.

R4 If had sent (TTP, β) to P_3^* on behalf of either P_1, P_2, P_4 in Round 3, send (y, TTP) to P_3^* on behalf of P_β.

$\mathcal{S}^1_{\mathsf{g4PC}}$ (P_1^* is corrupt)

R1 Simulation of Round 1 of $\mathcal{S}^1_{\mathsf{InputCommit}_\alpha}$ $(\alpha \in [4])$ (Figure 4.8). Let y denote the output computed.

R1 Receive s from P_1^* on behalf of P_2.

R1 Send pp_3 to P_1^* on behalf of P_3.

R2 Simulation of Round 2 of $\mathcal{S}^1_{\mathsf{InputCommit}_\alpha}$ $(\alpha \in [4])$ (Figure 4.8).

R2 On behalf of P_3: Receive $\mathcal{B}$ comprising of the garbled circuit, commitments on encoding and decoding information information and permutation strings p_{ij} for $(i \in [4], j \in \mathsf{ind}(\mathcal{P}_{i3}))$ from P_1^*. Additionally, the openings corresponding to the input labels x_{ij} for $(i \in [4], j \in \mathsf{ind}(\mathcal{P}_{i1}))$ (except the labels for x_{23}, x_{24}, x_{43}) are received.

R2 Following steps are executed: (a) Set $\mathcal{F}_3 = \{P_1, P_2\}$ if $\mathcal{B}$ is not consistent with $\mathcal{B}$ computed using randomness $\mathsf{G}(s)$ and pp_3, where s received on behalf of P_2 in Round 1. (b) If $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$, set P_1 to $\mathcal{C}_3$ if (openings of) encoding information for x_{ij}, for $i \in [4], j \in \mathsf{ind}(\mathcal{P}_{i3})$ are anything other than the originally committed labels (known on behalf of P_2). If any of the labels corresponding to $x_{ij} (i \in [4], j \notin \mathsf{ind}(\mathcal{P}_{i3}))$ do not correspond to the originally committed label (known on behalf of P_2), then set $\mathcal{F}_3 = \{P_1, P_2\}$. Here, x_{ij} refers to the value sent to P_1^* during $\mathsf{InputCommit}_i$ (for $i \in \mathsf{ind}(\mathcal{P}_1)$) on behalf of P_i or received on behalf of P_3 from P_1^* (during $\mathsf{InputCommit}_1$).

R2 Receive c from P_1^* on behalf of P_4. Add $\{P_1, P_2\}$ to $\mathcal{F}_4$ if c received from P_1^* is not consistent with $\mathcal{B}$ computed using s received on behalf of P_2.

R3 If $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$, compute Y such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$ (d known as simulator acts on behalf of P_2). Send Y to P_1^* on behalf of P_3.

R3 If $\mathcal{C}_\alpha \neq \emptyset \vee \mathcal{F}_\alpha \neq \emptyset$, P_α ($\alpha \in \{2,3,4\}$), send (TTP, β) to P_1^* where $P_\beta \notin \mathcal{C}_\alpha \cup \mathcal{F}_\alpha$.

R4 If Y was sent to P_1^* on behalf of P_3, send y to P_1^* on behalf of P_2, P_4.

R4 If received a valid view V_1 from P_1^* (comprising of openings corresponding to P_1's committed shares and the shares sent on behalf of honest parties in Round 1) along with (TTP, l), $l \in [4] \setminus \{1\}$ on behalf of P_l during Round 3, send (y, TTP) to P_1^* in Round 4 on behalf of P_l.

R4 If had sent (TTP, β) to P_1^* on behalf of either P_2, P_3, P_4 in Round 3, send (y, TTP) to P_1^* on behalf of P_β.

$\mathcal{S}^4_{\mathsf{g4PC}}$ **(P_4^* is corrupt)**

R1 Simulation of Round 1 of $\mathcal{S}^4_{\mathsf{InputCommit}_\alpha}$ ($\alpha \in [4]$) (Figure 4.8). Let y denote the output computed.

R2 Simulation of Round 2 of $\mathcal{S}^4_{\mathsf{InputCommit}_\alpha}$ ($\alpha \in [4]$) (Figure 4.8).

R2 Use uniform randomness to compute c as commitment on $\mathsf{H}(\mathsf{d})$. Send c to P_4^* on behalf of P_1, P_2.

R3 If $\mathcal{C}_g = \mathcal{F}_g = \emptyset$ for $P_g (g \in [2])$, send o (opening of hash of decoding information) to P_4^*.

R3 If $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$, compute Y such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$. Send Y to P_4^* on behalf of P_3.

R3 If $\mathcal{C}_\alpha \neq \emptyset \vee \mathcal{F}_\alpha \neq \emptyset$, P_α ($\alpha \in \{1,2,3\}$), send (TTP, β) to P_1^* where $P_\beta \notin \mathcal{C}_\alpha \cup \mathcal{F}_\alpha$.

R4 If Y was sent to P_4^*, send y to P_4^* on behalf of P_1, P_2.

R4 If received a valid view V_4 from P_4^* (comprising of openings corresponding to P_4's committed shares and the shares sent on behalf of honest parties in Round 1) along with (TTP, l), $l \in [4] \setminus \{1\}$ on behalf of P_l during Round 3, send (y, TTP) to P_4^* in Round 4 on behalf of P_l.

R4 If had sent (TTP, β) to P_4^* on behalf of either P_1, P_2, P_3 in Round 3, send (y, TTP) to P_4^* on behalf of P_β.

Figure 4.9: Description of $\mathcal{S}_{\mathsf{g4PC}}$

Security against corrupt P_3^*. We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}^3_{\mathsf{g4PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{g4PC},\mathcal{A}}$, when an adversary $\mathcal{A}$ corrupts P_3. The views are shown to be indistinguishable via a series of intermediate hybrids.

- HYB_0: Same as $\text{REAL}_{\mathsf{g4PC},\mathcal{A}}$.
- HYB_1: Same as HYB_0, except that when the execution does not result in P_3 getting access to the opening of commitment c_{ij} ($i \in \mathsf{ind}(\mathcal{P}_3), j \notin \mathsf{ind}(\mathcal{P}_{i3})$) sent by P_i, the commitment is replaced with commitment of dummy value.
- HYB_2: Same as HYB_1 except that P_3 is added to $\mathcal{C}_k$ ($k \in \mathsf{ind}(\mathcal{P}_3)$) if the opening forwarded by P_3 to P_k during $\mathsf{InputCommit}_i$ corresponding to P_i's committed share ($i \in \mathsf{ind}(\mathcal{P}_{3k})$) is anything other than what was originally committed.
- HYB_3: Same as HYB_2, except that P_1, P_2 use uniform randomness rather than pseudo-randomness.
- HYB_4: Same as HYB_3, except that some of the commitments of input wire labels sent on behalf of P_1, P_2, which will not be opened are replaced with commitments of dummy values.
- HYB_5: Same as HYB_4, except the following:
  - $\text{HYB}_{5.1}$: When the execution results in P_3 getting access to labels corresponding to its non-committed input for the garbled circuit, the GC is created as $(\mathsf{C}', \mathsf{X}) \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C)$ and the commitment to the hash of the decoding information is created for a dummy value.
  - $\text{HYB}_{5.2}$: When the execution results in P_3 getting access to labels corresponding to its committed input, the GC is created as $(\mathsf{C}', \mathsf{X}, d') \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y)$. The commitment c is computed on decoding information $\mathsf{H}(d')$.
- HYB_6: Same as HYB_5, except that P_3 does not receive y in Round 4 if neither P_1 nor P_2 receive Y obtained upon GC evaluation from P_3 in Round 3.
- HYB_7: Same as HYB_6 except that the TTP assigned by P_3 sends y only if the view V_3 sent by P_3 comprises of decommitments that opens to the input shares of the parties that were originally committed.

Since $\text{HYB}_7 := \text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}^3_{\mathsf{g4PC}}}$, we show that every two consecutive hybrids are computationally indistinguishable which concludes the proof.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that when the execution does not result in P_3 getting access to the opening of commitment c_{ij} ($i \in \mathsf{ind}(\mathcal{P}_3), j \notin \mathsf{ind}(\mathcal{P}_{i3})$) sent by P_i, c_{ij} corresponds to the actual input share x_{ij} in HYB_0 while it corresponds to dummy value in HYB_1. The indistinguishability follows from the hiding property of sCom.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is that while in HYB_1, P_3 is added to $\mathcal{C}_k$ ($k \in \mathsf{ind}(\mathcal{P}_3)$) if the opening forwarded by P_3 to P_k during $\mathsf{InputCommit}_i$ ($i \in \mathsf{ind}(\mathcal{P}_{3k})$) corresponding to P_i's committed share results in $\perp$; in HYB_2, $\mathcal{C}_k$ is set to P_3 if P_3 sends opening anything other than what was originally committed. Since the commitment scheme sCom is binding, in HYB_2, P_3 could have decommitted successfully to a different input share of P_i other than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is that P_1, P_2 use uniform randomness in HYB_3 rather than pseudorandomness as in HYB_2. The indistinguishability follows via reduction to the security of the PRG G.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The difference between the hybrids is that some of commitments of the input wire labels in HYB_3 that will not be opened are replaced with commitments of dummy values in HYB_4. The indistinguishability follows via reduction to the hiding property of the commitment scheme Com.

$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_{5.1}$: The difference between the hybrids is in the way (C, X) is generated when the execution results in P_3 getting access to labels corresponding to its non-committed input. In HYB_4, $(\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C')$ is run, which gives $(\mathsf{C}, \mathsf{En}(x, \mathsf{e}))$. In $\text{HYB}_{5.1}$, it is generated as $(\mathsf{C}', \mathsf{X}) \leftarrow \mathcal{S}_{\mathsf{obv}}(1^\kappa, C')$. Additionally, the commitment to the decoding information is created for a dummy value in $\text{HYB}_{5.1}$. The indistinguishability follows via reduction to the obliviousness of the garbling scheme and the hiding property of commitment scheme.

$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_{5.2}$: The difference between the hybrids is in the way $(\mathsf{C}, \mathsf{X}, \mathsf{d})$ is generated. In HYB_4, $(\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C')$ is run, which gives $(\mathsf{C}, \mathsf{En}(x, \mathsf{e}), \mathsf{d})$. In $\text{HYB}_{5.2}$, it is generated as $(\mathsf{C}', \mathsf{X}, \mathsf{d}') \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C', y)$. Additionally, the commitment to the decoding information is

computed on d'. The indistinguishability follows via reduction to the privacy of the garbling scheme and the hiding property of Com.

$\text{HYB}_5 \overset{c}{\approx} \text{HYB}_6$: The difference between the hybrids is that in HYB_5, P_3 does not receive y in Round 4 if neither P_1 nor P_2 receive Y such that $\mathsf{De}(\mathsf{Y}, d) \neq \perp$ from P_3; while in HYB_6, P_3 does not receive y if neither P_1 nor P_2 receive $\mathsf{Y} = \mathsf{Ev}(\mathsf{C}, \mathsf{X})$. Due to authenticity of the garbling scheme and the property of preimage-resistant hash used in the decoding information, P_3 could have sent Y such that $\mathsf{Y} \neq \mathsf{Ev}(\mathsf{C}, \mathsf{X})$ but $\mathsf{De}(\mathsf{Y}, d) \neq \perp$ only with negligibility probability. Therefore, the hybrids are indistiguishable.

$\text{HYB}_6 \overset{c}{\approx} \text{HYB}_7$: The difference between the hybrids is that in HYB_6, the TTP assigned by P_3 would return y to P_3 if the view V_3 sent by P_3 comprises of decommitments that lead to non-$\perp$ (corresponding to the commitments on shares output by the subroutine $\mathsf{InputCommit}$); while in HYB_7, the TTP assigned by P_3 would return y to P_3 only if the view V_3 sent by P_3 contains decommitments that open to the input shares that were originally committed. Since the commitment scheme sCom is (strong) binding even against an adversarially chosen pp; in HYB_6, P_3 could have decommitted successfully to a different input share than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

Security against corrupt P_1^*. We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^1_{\mathsf{g4PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{g4PC}, \mathcal{A}}$, when an adversary $\mathcal{A}$ corrupts P_1. The views are shown to be indistinguishable via a series of intermediate hybrids.

- HYB_0: Same as $\text{REAL}_{\mathsf{g4PC}, \mathcal{A}}$.
- HYB_1: Same as HYB_0, except that when the execution does not result in P_1 getting access to the opening of commitment c_{ij} ($i \in \mathsf{ind}(\mathcal{P}_1), j \notin \mathsf{ind}(\mathcal{P}_{i1})$) sent by P_i, the commitment is replaced with commitment of dummy value.
- HYB_2: Same as HYB_1 except that P_1 is added to $\mathcal{C}_k$ ($k \in \mathsf{ind}(\mathcal{P}_1)$) if the opening forwarded by P_1 to P_k during $\mathsf{InputCommit}_i$ corresponding to P_i's committed share ($i \in \mathsf{ind}(\mathcal{P}_{1k})$) is anything other than what was originally commited.
- HYB_3: Same as HYB_2, except that when $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$ at the end of Round 2, P_1 is added to $\mathcal{C}_3$ if P_3 receives anything other than the encoding information corresponding to committed share x_{ij} ($i \in [4], j \in \mathsf{ind}(\mathcal{P}_{i3})$).

– HYB_4: Same as HYB_3, except that when $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$ at the end of Round 2, $\{P_1, P_2\}$ is added to $\mathcal{F}_3$ if P_3 receives anything other than the encoding information corresponding to committed share x_{ij} $(i \in [4], j \notin \mathsf{ind}(\mathcal{P}_{i3}))$.

– HYB_5: Same as HYB_4, except that Y is computed via $\mathsf{De}(\mathsf{Y}, d) = y$ in place of $\mathsf{Y} = \mathsf{Ev}(\mathsf{C}, \mathsf{X})$.

– HYB_6: Same as HYB_5 except that the TTP assigned by P_1 sends y only if the view V_1 sent by P_1 comprises of decommitments that opens to the input shares of the parties that were originally committed.

Since $\text{HYB}_6 := \text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathbb{S}^1_{\mathsf{g4PC}}}$, we show that every two consecutive hybrids are computationally indistinguishable which concludes the proof.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that when the execution does not result in P_1 getting access to the opening of commitment $c_{ij} (i \in \mathsf{ind}(\mathcal{P}_1), j \notin \mathsf{ind}(\mathcal{P}_{i1}))$ sent by P_i, c_{ij} corresponds to the actual input share x_{ij} in HYB_0 while it corresponds to dummy value in HYB_1. The indistinguishability follows from the hiding property of sCom.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is that while in HYB_1, P_1 is added to $\mathcal{C}_k$ $(k \in \mathsf{ind}(\mathcal{P}_1))$ if the opening forwarded by P_1 to P_k during $\mathsf{InputCommit}_i$ $(i \in \mathsf{ind}(\mathcal{P}_{1k}))$ corresponding to P_i's committed share results in $\perp$; in HYB_2, $\mathcal{C}_k$ is set to P_1 if P_1 sends opening anything other than what was originally committed. Since the commitment scheme sCom is binding, in HYB_2, P_1 could have decommitted successfully to a different input share of P_i other than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is that in HYB_1, when $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$ at the end of Round 2, P_1 is added to $\mathcal{C}_3$ if the decommitments (corresponding to encoding of committed share x_{ij} $(i \in [4], j \in \mathsf{ind}(\mathcal{P}_{i3}))$) sent by P_1 output $\perp$ while in HYB_2, P_1 is added to $\mathcal{C}_3$ if the decommitments sent by P_1 open to any value other than the originally committed encoding information corresponding to x_{ij}. Since the commitment scheme Com is binding and pp was chosen uniformly at random by P_3; in HYB_1, P_1 could have decommitted successfully to a different input label than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The difference between the hybrids is that in HYB_3, when $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$ at the end of Round 2, $\{P_1, P_2\}$ is added to $\mathcal{F}_3$ if the index of the decommitments (corresponding

to encoding of committed share x_{ij} ($i \in [4], j \notin \mathsf{ind}(\mathcal{P}_{i3})$)) sent by P_1 are inconsistent with that known on behalf of P_2, while in HYB_4, $\{P_1, P_2\}$ is added to $\mathcal{F}_3$ if the decommitments sent by P_1 open to any value other than the originally committed encoding information corresponding to x_{ij}. Since the commitment scheme Com is binding and pp was chosen uniformly at random by P_3; in HYB_3, P_1 could have sent opening corresponding to the consistent index but decommitted successfully to a different input label than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_5$: The difference between the hybrids is that in HYB_4, Y is computed via $\mathsf{Ev}(\mathsf{C}, \mathsf{X})$, while in HYB_5, Y is computed such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$. Due to the correctness of the garbling scheme, the equivalence of Y computed via $\mathsf{Ev}(\mathsf{C}, \mathsf{X})$ or such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$ holds.

$\text{HYB}_5 \overset{c}{\approx} \text{HYB}_6$: The difference between the hybrids is that in HYB_5, the TTP assigned by P_1 would return y to P_1 if the view V_1 sent by P_1 comprises of decommitments that lead to non-$\perp$ (corresponding to the commitments on shares output by the subroutine $\mathsf{InputCommit}_i$); while in HYB_6, the TTP assigned by P_1 would return y to P_1 only if the view V_1 sent by P_1 contains decommitments that open to the input shares that were originally committed. Since the commitment scheme sCom is binding even against an adversarially chosen pp; in HYB_5, P_1 could have decommitted successfully to a different input share than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

Security against corrupt P_4^*. We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^4_{\mathsf{g4PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{g4PC}, \mathcal{A}}$, when an adversary $\mathcal{A}$ corrupts P_4. The views are shown to be indistinguishable via a series of intermediate hybrids.

- HYB_0: Same as $\text{REAL}_{\mathsf{g4PC}, \mathcal{A}}$.
- HYB_1: Same as HYB_0, except that when the execution does not result in P_4 getting access to the opening of commitment c_{ij} ($i \in \mathsf{ind}(\mathcal{P}_4), j \notin \mathsf{ind}(\mathcal{P}_{i4})$) sent by P_i, the commitment is replaced with commitment of dummy value.
- HYB_2: Same as HYB_1 except that P_4 is added to $\mathcal{C}_k$ ($k \in \mathsf{ind}(\mathcal{P}_4)$) if the opening forwarded by P_4 to P_k during $\mathsf{InputCommit}_i$ corresponding to P_i's committed share ($i \in \mathsf{ind}(\mathcal{P}_{4k})$) is anything other than what was originally committed.
- HYB_3: Same as HYB_2, except that P_1, P_2 use uniform randomness rather than pseudo-randomness.

– HYB_4: Same as HYB_4, except that Y is computed via $\mathsf{De}(\mathsf{Y}, d) = y$ in place of $\mathsf{Y} = \mathsf{Ev}(\mathsf{C}, \mathsf{X})$.

– HYB_5: Same as HYB_4 except that the TTP assigned by P_4 sends y only if the view V_4 sent by P_4 comprises of decommitments that opens to the input shares of the parties that were originally committed.

Since $\text{HYB}_5 := \text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^4_{\mathsf{g4PC}}}$, we show that every two consecutive hybrids are computationally indistinguishable which concludes the proof.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that when the execution does not result in P_4 getting access to the opening of commitment c_{ij} ($i \in \mathsf{ind}(\mathcal{P}_4), j \notin \mathsf{ind}(\mathcal{P}_{i4})$ sent by P_i, c_{ij} corresponds to the actual input share x_{ij} in HYB_0 while it corresponds to dummy value in HYB_1. The indistinguishability follows from the hiding property of sCom.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is that while in HYB_1, P_4 is added to $\mathcal{C}_k$ ($k \in \mathsf{ind}(\mathcal{P}_1)$) if the opening forwarded by P_4 to P_k during $\mathsf{InputCommit}_i$ ($i \in \mathsf{ind}(\mathcal{P}_{4k})$) corresponding to P_i's committed share results in $\perp$; in HYB_2, $\mathcal{C}_k$ is set to P_4 if P_4 sends opening anything other than what was originally committed. Since the commitment scheme sCom is binding, in HYB_2, P_4 could have decommitted successfully to a different input share of P_i other than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is that P_1, P_2 use uniform randomness in HYB_1 rather than pseudorandomness as in HYB_1. The indistinguishability follows via reduction to the security of the PRG G.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The difference between the hybrids is that in HYB_3, Y is computed via $\mathsf{Ev}(\mathsf{C}, \mathsf{X})$, while in HYB_4, Y is computed such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$. Due to the correctness of the garbling scheme, the equivalence of Y computed via $\mathsf{Ev}(\mathsf{C}, \mathsf{X})$ or such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$ holds.

$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_5$: The difference between the hybrids is that in HYB_4, the TTP assigned by P_4 would return y to P_4 if the view V_4 sent by P_4 comprises of decommitments that lead to non-$\perp$ (corresponding to the commitments on shares output by the subroutine $\mathsf{InputCommit}$); while in HYB_5, the TTP assigned by P_4 would return y to P_4 only if the view V_4 sent by P_4 contains decommitments that open to the input shares that were originally committed. Since the commitment scheme sCom is binding even against an adversarially chosen pp; in HYB_4,

P_4 could have decommitted successfully to a different input share than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable. This completes the proof.

4.8.3 Security Proof for g4PC4

In this section, we provide a high-level overview of the proof of Theorem 4.5 that states the security of g4PC4 relative to its ideal functionality.

We describe the simulator $\mathcal{S}_{\mathsf{g4PC4}}$ for the cases of corrupt P_1 and P_3. The corruption of P_2 and P_4 is analogous to the case of P_1 and P_3 respectively. We give only a sketch of the simulator below since the simulation proceeds almost exactly as the simulation of g4PC described formally in Section 4.8.2.

For the case when P_3 is corrupt, simulator $\mathcal{S}^3_{\mathsf{g4PC4}}$ acts on behalf of honest P_1, P_2, P_4 as follows: In round 1 of $\mathsf{InputCommit}_\alpha, \alpha \in \mathcal{P}_3$, $\mathcal{S}^3_{\mathsf{g4PC4}}$ chooses random values corresponding to the shares of honest parties accessible to P_3, namely x_{ij} $(i \in \mathcal{P}_3, j \in \mathcal{P}_{i3})$ and acts according to the protocol. Commitments on the remaining shares of honest parties are dummy. Correspondingly, on behalf of the honest parties, simulator receives commitments corresponding to $x_{3j} (j \in \mathcal{P}_3)$ in round 1 of $\mathsf{InputCommit}_3$ and checks if there exists a majority commitment corresponding to each of the shares. If not, P_3 is added to $\mathcal{C}_i$ $(i \in \mathcal{P}_3)$ and $\mathcal{F}_{\mathsf{god}}$ is invoked with default value to retrieve y. Else, P_3's input is extracted using the shares corresponding to the majority commitment and its opening. Consequently, $\mathcal{F}_{\mathsf{god}}$ is invoked using the committed input of P_3 and y is obtained. The corrupt and conflict sets of the honest parties are populated according to the protocol. For simulation of Round 2 on behalf of garblers, we consider two cases depending on whether: (a) P_3 gets access to the labels corresponding to any of its non-committed input shares (b) P_3 gets access to labels corresponding to its committed input shares. The case that will follow can be determined at the end of Round 1 itself by simulator acting on behalf of the honest garblers since P_3's committed input is known to simulator by then. Accordingly in Round 2, either the oblivious simulator of the garbling scheme $\mathcal{S}_{\mathsf{obv}}$ or the privacy simulator $\mathcal{S}_{\mathsf{prv}}$ (can be invoked with output y obtained) is invoked for case (a) and (b) respectively. In case (a) when GC returned by $\mathcal{S}_{\mathsf{obv}}$ is used, the commitment on hash of the decoding information is dummy and never has to be opened to P_3 according to the protocol steps as for each garbler P_g atleast one of $\mathcal{C}_g \neq \emptyset / \mathcal{F}_g \neq \emptyset$ holds. In the latter case when GC returned by $\mathcal{S}_{\mathsf{prv}}$ is used, the commitment is done on the value $\mathsf{H}(d)$, where d is returned by $\mathcal{S}_{\mathsf{prv}}$. This commitment is opened during Round 3 by simulator acting on behalf of garbler P_g if $\mathcal{C}_g = \mathcal{F}_g = \emptyset$.

Next, if P_3 sends a (TTP, β) message to a party in $\mathcal{P}_3$ and sends a valid V_3 (with openings

of committed shares known to simulator) to P_β in Round 3, (y, TTP) is sent to P_3 on behalf of P_β. Additionally, the conflict and corrupt set of $P_i, i \in \mathcal{P}_3$ that are locally computed (during simulation of $\mathsf{InputCommit}_3$) are used by P_i to identify TTP as per the protocol and (TTP, β) message is sent accordingly to P_3 in Round 3. Subsequently, (y, TTP) is sent to P_3 on behalf of the P_β.

For the case when P_1 is corrupt, simulator $\mathcal{S}^1_{\mathsf{g4PC4}}$ acts on behalf of P_2, P_3, P_4 as follows: Simulation of $\mathsf{InputCommit}_\alpha, \alpha \in [4]$ is the same as described for $\mathcal{S}^3_{\mathsf{g4PC4}}$ which would lead to extraction of P_1's committed input and retrieval of y via $\mathcal{F}_{\mathsf{god}}$. On behalf of the evaluator, say P_3 if $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$ (populated during simulation of executions of $\mathsf{InputCommit}_\alpha()$), the simulator checks if P_1 (a) sends GC consistent with randomness shared with P_2 (b) sends encoding of committed input shares. If either of the checks fails, the corrupt or conflict set of P_3 is populated accordingly (for (b), incase of shares known to P_3, corrupt set is populated; else conflict is populated with $\{P_1, P_2\}$ corresponding to shares that are not held by P_3 and held by both garblers) and the TTP is assigned as per the protocol. The output y is sent to P_1 on behalf of the TTP in Round 4. If the checks pass and $\mathcal{C}_i = \mathcal{F}_i = \emptyset, i \in \{3, 4\}$, then Y is computed such that it decodes to output y and sent to P_1 on behalf of P_i in round 3. This completes the simulation sketch of $\mathsf{g4PC4}$.

4.8.4 Security Proof for protocol g3PC

In this section, we present the proof of Theorem 4.7 that states the security of GOD relative to its ideal functionality. We describe the simulator $\mathcal{S}_{\mathsf{g3PC}}$ for the case when P_1, P_3 is corrupt. The case of P_2 being corrupt is symmetric to that of P_1. The description of the simulator is available in Figure 4.10 with **R1/R2/R3/R4/R5** indicating simulation for round 1, 2, 3, 4 and 5 respectively.

Simulator $\mathcal{S}_{\mathsf{g3PC}}$

$\mathcal{S}^3_{\mathsf{g3PC}}$ **(P_3^* is corrupt)**

R1 Receive $(\mathsf{pp}, c_{31}, c_{32})$ via broadcast and $(x_{31}, x_{32}, \mathsf{o}_{31}, \mathsf{o}_{32})$ privately from P_3^* on behalf of P_1, P_2.

R2 Broadcast $(\texttt{Conflict}, P_3)$ on behalf of P_i if $\mathsf{Open}(c_{3i}, \mathsf{o}_{3i}) \neq x_{3i}$ for $i \in [2]$. If for exactly one i the check doesn't pass, set TTP to $P_{[2]\setminus i}$ and broadcast $\mathcal{B}$ as per protocol on behalf of $P_{[2]\setminus i}$. If check doesn't pass for both $i \in [2]$, set $\mathsf{TTP} = P_1$.

R2 If $\mathsf{TTP} = \emptyset$, extract $x_3 = x_{31} \oplus x_{32}$ and invoke $\mathcal{F}_{\mathsf{god}}$ with (Input, x_3) on behalf of P_3^* to retrieve output y. Use uniform randomness r on behalf of P_1, P_2 and run $(\mathsf{C}, \mathsf{X}, \mathsf{d}) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y)$.

R2 If $\mathsf{TTP} = \emptyset$, choose m_1, m_2 at random. Let $\{c_\alpha^{m_1^\alpha}, c_{\ell+\alpha}^{m_2^\alpha}, c_{2\ell+\alpha}^{x_{31}^\alpha}, c_{3\ell+\alpha}^{x_{32}^\alpha}\}_{\alpha\in[\ell]}$ be commitments to the entries of X. Commit to dummy values corresponding to other input labels. Set $\mathcal{B}$ to include C and set of commitments. Broadcast $\mathcal{B}$ on behalf of both P_1, P_2. Send $(\{\mathrm{o}_\alpha^{m_1^\alpha}, \mathrm{o}_{2\ell+\alpha}^{x_{31}^\alpha}\}_{\alpha\in[\ell]}, m_1)$, $(\{\mathrm{o}_{\ell+\alpha}^{m_2^\alpha}, \mathrm{o}_{3\ell+\alpha}^{x_{32}^\alpha}\}_{\alpha\in[\ell]}, m_2)$ on behalf of P_1, P_2 respectively.

R3 If received broadcast of $(\texttt{Conflict}, P_i)(i \in [2])$ from P_3, set $\mathsf{TTP} = P_{[2]\setminus i}$. Else if received $\mathsf{Y} \neq (\mathsf{C}, \mathsf{X})$, set $\mathsf{TTP} = P_1$.

R4 If $\mathsf{TTP} \neq \emptyset$ and $y = \perp$: Invoke $\mathcal{F}_{\mathsf{god}}$ with (Input, x_3) to get output y where x_3 is computed using $\mathrm{o}_{31}, \mathrm{o}_{32}$ received in Round 1 on behalf of honest parties, else received from P_3^* on behalf of TTP in Round 4 (take default value if not received or invalid).

R5 Send y to P_3^* on behalf of TTP if $\mathsf{TTP} \neq \emptyset$.

$\underline{\mathcal{S}^1_{\mathsf{g3PC}}\ (P_1^*\ \textbf{is corrupt})}$

R1 Choose x_{31}, pp at random. Compute $(c_{31}, \mathrm{o}_{31}) \leftarrow \mathsf{Com}(\mathsf{pp}, x_{31})$. Broadcast $\{\mathsf{pp}, c_{31}, c_{32}\}$ where c_{32} is commitment of dummy value. Send $\{x_{31}, \mathrm{o}_{31}\}$ to P_1^* on behalf of P_3.

R2 Compute and broadcast $\mathcal{B}_2$ on behalf of P_2, using s received from P_1^* as per protocol.

R2 Set $\mathsf{TTP} = P_3$ if $\mathcal{B}_1 \neq \mathcal{B}_2$. Set $\mathsf{TTP} = P_2$ if P_1 broadcasts $(\texttt{Conflict}, P_3)$

R3 Suppose $\mathsf{TTP} = \emptyset$: Check if any of the decommitments sent by P_1^* to P_3 in Round 2 opens to something other than what was originally committed (known on behalf of P_2). If so, broadcast $(\texttt{Conflict}, P_1)$ on behalf of P_3 and set $\mathsf{TTP} = P_2$.

R3 If $\mathsf{TTP} = \emptyset$, extract P_1^*'s input as $x_1 = m_1 \oplus p_1$, where p_1, m_1 is known on behalf of P_2, P_3 respectively. Invoke $\mathcal{F}_{\mathsf{god}}$ with (Input, x_1) to receive output y. Compute Y such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$ (d known to P_2) and broadcast Y on behalf of P_3.

R4 If $\mathsf{TTP} \neq \emptyset$, receive x_1 from P_1^* (take default value if not received) on behalf of TTP. Invoke $\mathcal{F}_{\mathsf{god}}$ with (Input, x_1) to retrieve output y.

R5 If $\mathsf{TTP} \neq \emptyset$, send y to P_1^* on behalf of TTP.

Figure 4.10: Description of $\mathcal{S}_{\mathsf{g3PC}}$

Security against corrupt P_3^*. We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^3_{\mathsf{g3PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{g3PC}, \mathcal{A}}$, when $\mathcal{A}$ corrupts P_3. The views are shown to be indistinguishable via a series of intermediate hybrids.

– HYB_0: Same as $\text{REAL}_{\mathsf{g3PC}, \mathcal{A}}$.

– HYB_1: Same as HYB_0, except that P_1, P_2 use uniform randomness rather than pseudorandomness.

– HYB_2: Same as HYB_1, except that some of the commitments of input wire labels sent by P_1, P_2, which will not be opened are replaced with commitments of dummy values. Specifically, these are the commitments with indices $\neq m_1, m_2, x_{31}, x_{32}$.

– HYB_3 : Same as HYB_2, except that when the execution results in P_3 evaluating the garbled circuit (GC), the GC is created as $(\mathsf{C}', \mathsf{X}, d') \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y)$.

– HYB_4 : Same as HYB_3, except that P_1 is set to TTP if P_3 broadcasts $\mathsf{Y} \neq (\mathsf{C}, \mathsf{X})$.

Since $\text{HYB}_4 := \text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^3_{\mathsf{g3PC}}}$, we show that every two consecutive hybrids are computationally indistinguishable which concludes the proof.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that P_1, P_2 use uniform randomness in HYB_1 rather than pseudorandomness as in HYB_0. The indistinguishability follows via reduction to the security of the PRG G.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is that some of commitments of the input wire labels in HYB_1 that will not be opened are replaced with commitments of dummy values in HYB_2. The indistinguishability follows via reduction to the hiding property of the commitment scheme Com that holds even though pp was chosen by corrupt P_3.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is in the way $(\mathsf{C}, \mathsf{X}, \mathsf{d})$ is generated. In HYB_2, $(\mathsf{C}, \mathsf{e}, \mathsf{d}) \leftarrow \mathsf{Gb}(1^\kappa, C)$ is run, which gives $(\mathsf{C}, \mathsf{En}(x, \mathsf{e}), \mathsf{d})$. In HYB_3, it is generated as $(\mathsf{C}', \mathsf{X}, \mathsf{d}_1) \leftarrow \mathcal{S}_{\mathsf{prv}}(1^\kappa, C, y)$. The indistinguishability follows via reduction to the privacy of the garbling scheme.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The difference between the hybrids in that while in HYB_3, P_1 is set to TTP when P_3 broadcasts Y such that $\mathsf{De}(\mathsf{Y}, d) = \perp$; in HYB_4, P_1 is set to TTP when P_3 broadcasts $\mathsf{Y} \neq (\mathsf{C}, \mathsf{X})$. It follows from the authenticity property of garbling that P_3 will be able to come up with Y such that $\mathsf{Y} \neq (\mathsf{C}, \mathsf{X})$ but $\mathsf{De}(\mathsf{Y}, d) \neq \perp$ only with negligible probability.

Security against corrupt P_1^*. We now argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^1_{\mathsf{g3PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{g3PC}, \mathcal{A}}$, when $\mathcal{A}$ corrupts P_1. The views are shown to be indistinguishable via a series of intermediate hybrids.

– HYB_0: Same as $\text{REAL}_{\mathsf{g3PC},\mathcal{A}}$

– HYB_1: Same as HYB_0, except that P_3 raises a conflict with P_1 if it accepts any decommitment that opens to a value other than what was originally committed.

– HYB_2: Same as HYB_1, except that when the execution does not result in P_1 getting access to the opening of commitment c_{32} (corresponding to x_{32}) broadcast by P_3, the commitment is replaced with commitment of dummy value.

– HYB_3: Same as HYB_2, except that Y is computed via $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$ rather than $\mathsf{Y} = \mathsf{Ev}(\mathsf{C}, \mathsf{X})$

Since $\text{HYB}_3 := \text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^1_{\mathsf{g3PC}}}$, we show that every two consecutive hybrids are computationally indistinguishable which concludes the proof.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that in HYB_0, P_3 raises a conflict with P_1 if the decommitments sent by P_1 output $\perp$ while in HYB_1, P_3 raises a conflict if the decommitments sent by P_1 opens to any value other than what was originally committed. Since the commitment scheme Com is binding and pp was chosen uniformly at random by P_3; in HYB_1, P_1 could have decommitted successfully to a different input label than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference between the hybrids is that when the execution does not result in P_1 getting access to the opening of commitment c_{32} (corresponding to x_{32}) broadcast by P_3, c_{32} corresponds to the actual input share x_{32} in HYB_1 while it corresponds to dummy value in HYB_2. The indistinguishability follows from the hiding property of Com.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is that in HYB_2, P_3 computes Y via $\mathsf{Ev}(\mathsf{C}, \mathsf{X})$, while in HYB_3, Y is computed such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$. Due to the correctness of the garbling scheme, the equivalence of Y computed via $\mathsf{Ev}(\mathsf{C}, \mathsf{X})$ or such that $\mathsf{De}(\mathsf{Y}, \mathsf{d}) = y$ holds.

Part II

Round Complexity of MPC : Beyond Traditional Adversaries

Chapter 5

On the Round Complexity of Best-of-Both-Worlds Multi-party Computation

In this chapter, we study the exact round complexity of 'Best of Both Worlds (BoBW)' MPC protocols, a much sought-after species of MPC protocols that offer the best possible security depending on the actual corruption. Specifically, such protocols simultaneously provide fn / god in honest majority setting and ua in dishonest majority setting.

5.1 Introduction

While highly sought-after, fairness (fn) and guaranteed output delivery (god) can only be realised, when majority of the involved population is honest [65]. In the absence of this favorable condition, uananimous abort (ua) is the best security notion that can be attained. With these distinct affordable goals, MPC with honest majority [30, 56, 177, 19, 17, 72, 4] and dishonest majority [107, 73, 94, 44, 3, 113, 15] mark one of the earlier demarcations in the world of MPC. While the constructions of each type are abound in the literature, one class of protocols does not seem to withstand the threat model of the other. For instance, the honest-majority protocols do not guarantee privacy of the inputs of the honest parties in the face of dishonest majority and likewise the dishonest-majority protocols cannot achieve god and fn, tolerating even a single corruption, let alone dishonest minority. In many real-life scenarios, it is highly unlikely for anyone to guess upfront how many parties the adversary is likely to corrupt. In such a scenario, the best a practitioner can do, is to employ the 'best' protocol from her favorite class and hope that the adversary will be within assumed corruption limit of the employed

protocol. If the guess fails, the employed protocol, depending on whether it is an honest or dishonest majority protocol, will suffer from the above mentioned issues. To overcome this worrisome shortcoming, an unconventional yet much sought-after species of MPC, termed as 'Best-of-Both-Worlds' (BoBW) which offers the best possible security depending on the actual corruption scenario was introduced in [124, 134, 127].

Denoting the threshold of corruption in honest and dishonest majority case by t and s respectively, an ideal BoBW MPC should promise the best possible security in each corruption scenario for any population of size n, as long as $t < n/2$ and $s < n$. Quite contrary to the expectation, the grand beginning of BoBW MPC with the works of [124, 134, 127] is mostly marred with pessimistic results showing the above goal is impossible for many scenarios. For reactive functionalities that receive inputs and provide outputs in multiple rounds maintaining a state information between subsequent invocations, it is impossible to achieve BoBW security [124]. While theoretical feasibility is not declined, non-reactive or standard functionalities are shown to be impossible to realise as long as $t + s \geq n$ in expected polynomial time (in the security parameter), making any positive result practically irrelevant [134, 127]. A number of meaningful relaxations were proposed in the literature to get around the impossibility of BoBW security when $t + s \geq n$ [134, 127]. The most relevant to our work is the relaxation proposed in [152] where the best possible security of god is compromised to the second-best notion of fn in the honest-majority setting. Other attempts to circumvent the impossibility result appear in [124] and [134, 24] where the security in dishonest-majority setting is weakened to allowing the adversary to learn s evaluations of the function (each time with distinct inputs *exclusively* corresponding to the corrupt parties) in the former and achieving a weaker notion of $O(1/p)$-security with abort (actions of any polynomial-time adversary in the real world can be simulated by a polynomial-time adversary in the ideal world such that the distributions of the resulting outcomes cannot be distinguished with probability better than $O(1/p)$) in the latter. [124] shows yet another circumvention by weakening the adversary in dishonest-majority case from active to passive. On the contrary, constructions are known when $t + s < n$ is assumed [124], tolerating active corruptions and giving best possible security in both the honest and dishonest majority case.

In this work, we consider two types of BoBW MPC protocols and study their exact round complexity: (a) MPC achieving the best security of god and ua in the honest and dishonest majority setting respectively assuming $s+t < n$, referred as ($\mathsf{god}|\mathsf{ua}$)-BoBW; (b) MPC achieving second-best security notion of fn in the honest majority and the best possible security of ua in the dishonest majority for any n, referred as ($\mathsf{fn}|\mathsf{ua}$)-BoBW. We nearly settle the exact round complexity of these two classes under the assumption of no setup (plain model), public setup

(CRS) and private setup (CRS + PKI or simply PKI).

5.1.1 Related Work

We refer to Section 1.3 for relevant literature on exact round complexity of MPC in the traditional settings of honest and dishonest majority and outline works related to BoBW MPC below. An orthogonal notion of BoBW security is considered in [54, 119, 152] where information-theoretic and computational security is the desired goal in honest and dishonest majority setting respectively. Avoiding the relaxation to computational security in dishonest-majority setting, the work of [114] introduces the best possible information-theoretic guarantee achievable in the honest and dishonest majority settings simultaneously; i.e the one that offers standard information-theoretic security in honest majority and offers residual security (the adversary cannot learn anything more than the residual function of the honest parties' inputs) in dishonest-majority setting. A more fine-grained graceful degradation of security is dealt with in the works of [152, 120, 121, 122, 169] considering a mixed adversary that can simultaneously corrupt in both active and semi-honest style. [99] studies the communication efficiency in the BoBW setting. In spite of immense practical relevance of BoBW protocols, the question of their exact round complexity has not been tackled so far. Constant-round protocols are presented in (or can be derived from) [124, 127] for (god|ua)-BoBW and BoBW where only semi-honest corruptions are tolerated in the dishonest majority. The recent work of [169] settled the exact round complexity of the latter class, as a special case of a strong adversarial model that allows both active (with threshold t_a) and passive (with threshold t_p, which subsumes the active corruptions) corruption for a range of thresholds for (t_a, t_p) starting from $(\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ to $(0, n-1)$. Lastly, the round complexity of BoBW protocols of [24] that achieve $1/p$-security with abort in dishonest majority (and god in honest majority), depends on the polynomial $p(\kappa)$ (where κ denotes the security parameter).

5.1.2 Our Results

Assuming a network with pair-wise private channels and a broadcast channel, we show that 5 and 3 rounds are necessary and sufficient for (fn|ua)-BoBW MPC under the assumption of 'no setup' and 'public and private setup' respectively. For the class of (god|ua)-BoBW MPC, we show necessity and sufficiency of 3 rounds for the public setup case and 2 rounds for the private setup case. In the no setup setting, we show the sufficiency of 5 rounds, while the known lower bound is 4. All our upper bounds are based on polynomial-time assumptions and assume black-box simulation. With distinct feasibility conditions, the classes differ in terms of the round requirement. The bounds are in some cases different and on a positive note at most one

more, compared to the maximum of the needs of the honest-majority and dishonest-majority setting. While the details of our results appear in Section 1.4.2, we summarise and put them along with the bounds known in the honest and dishonest majority setting in Table 5.1 for quick reference.

	No setup (Plain Model)	**Public Setup (CRS)**	**Private Setup (CRS + PKI)**
Honest Majority $t < n/2$ fn / god	Round: 3 Lower Bound: [166, 102] Upper Bound: [4, 16]	Round: 3 Lower Bound: [166, 102] Upper Bound: [108, 4, 16]	Round: 2 Lower Bound: [112] Upper Bound: [108]
Dishonest Majority $s < n$ sa / ua	Round: 4 Lower Bound: [95] Upper Bound: [113, 15, 60] (sa only)	Round: 2 Lower Bound: [112] Upper Bound: [94, 160] [92, 93, 35]	Round: 2 Lower Bound: [112] Upper Bound: [94, 160] [92, 93, 35]
(fn|ua)-**BoBW** $t < n/2, s < n$ fn & ua	Round: 5 Lower Bound: **This work** Upper Bound: **This work**	Round: 3 Lower Bound: [102, 166] Upper Bound: **This work**	Round: 3 Lower Bound: **This work** Upper Bound: **This work**
(god|ua)-**BoBW** $t < n/2, t + s < n$ god & ua	Round: – Lower Bound: 4 [95] Upper Bound: 5 **This work**	Round: 3 Lower Bound: **This work** Upper Bound: **This work**	Round: 2 Lower Bound: [112] Upper Bound: **This work**

Table 5.1: Summary of results related to BoBW MPC

Extensions. We can boost the security of all our protocols to offer identifiability (i.e. public identifiability of the parties who misbehaved) when abort happens– (fn|ua)-BoBW protocols with identifiable fairness (idfair) and abort (idua) in honest and dishonest majority setting respectively and (god|ua)-BoBW protocols with idua in dishonest-majority setting. Our lower bound results hold as is when ua and fn are upgraded to their stronger variants with identifiability. Furthermore, all our upper bounds relying on CRS have instantiations based on a weaker setup, referred as common *random* string, owing to the availability of 2-round Oblivious Transfer (OT) [176] and Non-Interactive Zero Knowledge (NIZK) [179] under the latter setup assumption. Lastly, we also propose few optimizations to minimize the use of broadcast channels in our compilers upon which our upper bounds are based. Specifically, these optimizations preserve the round complexity of our upper bounds at the cost of relaxing the security notion in dishonest majority setting to sa (as opposed to ua).

5.1.3 Techniques

(fn|ua)-BoBW. The lower bounds are obtained via a reduction to 3-round OT in plain model and 1-round OT in private setup setting, both of which are known to be impossible [95, 112]

(albeit under the black-box simulation paradigm which is of concern in this work). The starting point is a protocol π between 3 parties which provides fn when 1 party is corrupt and ua when 2 parties are corrupt, in 4 rounds when no setup is assumed and 2 rounds when private/public setup is assumed. The heart of the proof lies in devising a function f such that the realization of f via π, barring its last round, leads to an OT.

The upper bounds are settled with a proposed generic compiler that turns an r-round dishonest-majority MPC protocol achieving ua to an $(r+1)$-round BoBW MPC protocol *information-theoretically*. The compiler churns out a 5-round and a 3-round BoBW protocol in the plain model and in the presence of a CRS respectively, when plugged with appropriate ua-secure dishonest-majority protocol in the respective setting. Since the constructions of the known 4-round dishonest-majority MPC relying on polynomial-time assumptions [113, 15, 60] provide only sa security, we transform them to achieve ua for our purpose which invokes non-triviality for [113]. With CRS, the known constructions of [93, 35] achieve unanimity and readily generate 3-round BoBW protocols.

Our compiler motivated by [130] uses the underlying r-round protocol to compute authenticated secret sharing of the output y with a threshold $t(< n/2)$ enabling the output reconstruction to occur in the last round. Fairness is ensured given the unanimity of the underlying protocol and the fact that the adversary (controlling t corrupt parties) has no information about the output y from the t shares he owns. However, using pairwise MACs for authentication defies unanimity in case of arbitrary corruptions because a corrupt party can choose to provide a verified share to a selected set of honest parties enabling their output reconstruction while causing the rest to abort. This issue is tackled by enforcing public verifiability of the shares via a form of authentication used in the Information Checking Protocol (ICP) primitive of [173, 165] and unanimously identifiable commitments (UIC) of [128]. This technique makes it impossible (except with negligible probability) for a corrupt party to keep two honest parties on different pages about the correctness of the share it provides during output reconstruction hence preserving unanimity.

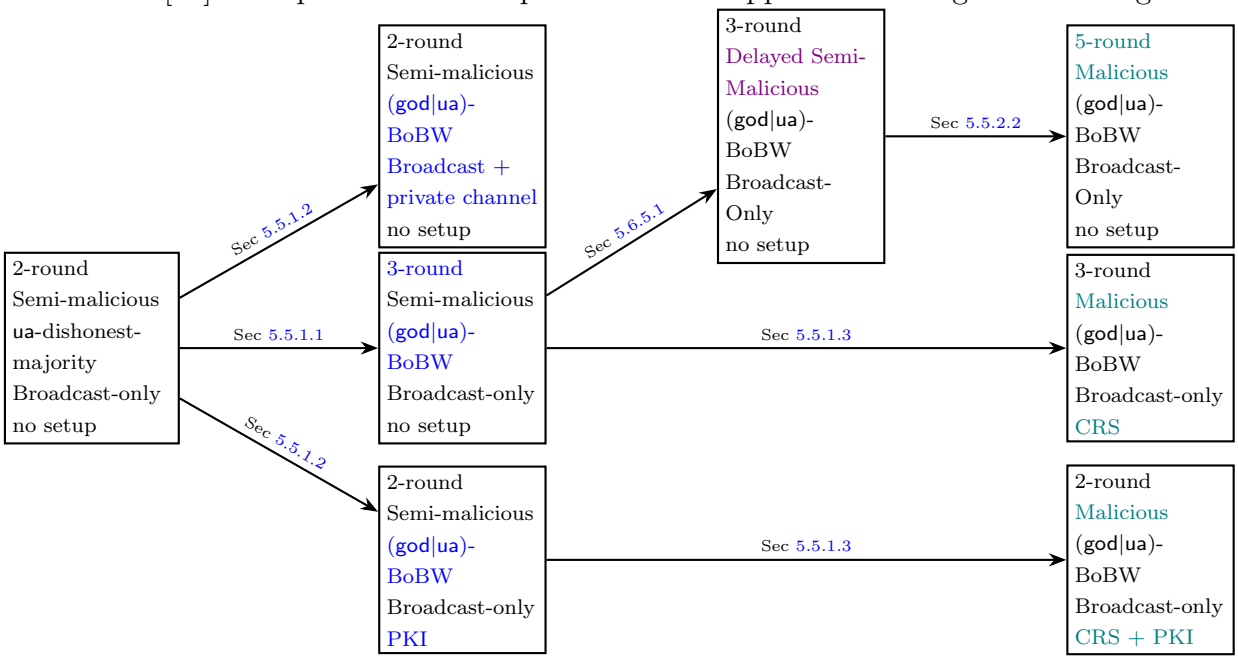
(god|ua)-BoBW. The non-trivial lower bound for this class is for the CRS setting. The other bounds imply from the dishonest-majority case. In the CRS setting, we prove a lower bound of 3 rounds. We start with assuming a 2 round BoBW protocol π for a specifically articulated 4-party function f. Next, we consider a sequence of executions of π, with different adversarial strategies in the order of their increasingly malicious behaviour such that the views of a certain party stays the same between the executions. This sequence finally leads us to a strategy where the adversary is able to learn the input of an honest party breaching privacy, hence coming to a contradiction. The crux of the lower bound argument lies in the design

of the adversarial strategies that shuffle between the honest and dishonest majority setting encapsulating the challenge in designing BoBW protocols. This is in contrast to existing lower bounds in traditional models that deal with a fixed setting and single security notion at a time.

In the presence of a CRS, we build a 3-round protocol in two steps: a) we provide a generic compiler that transforms a broadcast-only ua-secure 2-round semi-malicious protocol such as [93, 35] to a 3-round broadcast-only BoBW protocol of this class against a semi-malicious adversary (that follows the protocol honestly but can choose bad random coins for each round which are available to the simulator) b) then, the round-preserving compiler of [10] (using NIZKs) is applied on the above protocol to attain malicious security. The first compiler, in spirit of [4], ensures god against t non-cooperating corrupt parties in the last round, via secret-sharing the last-round message of the underlying protocol during the penultimate round of the compiled protocol. This is achieved by means of a garbled circuit sent by each party outputting its last-round message of the underlying protocol and the shares of the encoded labels with a threshold of s so that $s+1$ parties (in case of honest majority) can come together in the final round to construct the last-round message of the corrupt parties. This garbled circuit of a party P_i also takes into account the case when some other parties abort in the initial rounds of the protocol by taking the list of aborting parties as input and hard-coding their default input and randomness such that P_i's last round message is computed considering default values for parties who aborted. The compiler is made round-preserving with the additional provision of pairwise-private channels or alternately, PKI. The latter (with PKI) just like its 3-round avatar can be compiled to a malicious protocol via the compiler of [10].

In the plain model, we provide a 5-round construction which is substantially more involved than our other upper bounds. To cope up with the demands of (god|ua)-BoBW security in the plain model, we encountered several roadblocks that were addressed by adapting some existing techniques combined with new tricks. The construction proceeds in two steps: a) we boost the security of our broadcast-only 3-round semi-malicious BoBW protocol to a stronger notion of delayed-semi-malicious security (where the adversary is required to justify his messages by giving a valid witness only in the last but one round) and b) we plug this 3-round BoBW protocol in the compiler of [35] with some additional modifications to obtain a 5-round BoBW protocol secure against a malicious adversary. The compiler of [35] takes as input a $(k-1)$-round protocol secure *with abort* against a delayed-semi-malicious adversary and churns out a k-round protocol secure *with abort* against a malicious adversary for any $k \geq 5$. The major challenges in our construction surface in simulation, where we cannot terminate in the honest-majority case even if the adversary aborts on behalf of a corrupt party (unlike the compiler of [35] that achieves abort security only). Furthermore, we observed that the natural simulation

strategy to retain the BoBW guarantee suffered from a subtle flaw, similar to the one pointed in the work of [115], which we resolve with the help of the idea suggested therein. To bound the simulation time by expected polynomial-time, we further needed to introduce two 'dummy' rounds (rounds which do not involve messages of the underlying protocol being compiled) in our compiler as opposed to one as in [35]. This does not inflate the round complexity as our underlying delayed-semi-malicious protocol only consumes 3 rounds (instead of 4 as in the case of [35]). As a step towards resolving the question left open in this work (namely proving the impossibility or alternately constructing a 4-round (god|ua)-BoBW protocol under polynomial-time assumption), we present a sketch of a 4-round (god|ua)-BoBW protocol based on sub-exponentially secure trapdoor permutations and ZAPs. This construction builds upon the work of [64]. The pictorial roadmap to obtain the upper bounds is given in the figure below.



Model. Before moving onto the technical section, we detail our model here. We consider a set of n PPT parties $\mathcal{P} = \{P_1, \ldots P_n\}$ connected by pairwise-secure and authentic channels and having access to a broadcast channel. A few protocols in our work that are referred to as being *broadcast-only* do not assume private channels. Each party is modelled as a probabilistic polynomial time (PPT) Turing machine. We assume that there exists a PPT adversary $\mathcal{A}$, who can corrupt a subset of these parties.

Roadmap. Our lower and upper bounds for (fn|ua)-BoBW appear in Section 5.2-5.3. Our lower and upper bounds for (god|ua)-BoBW appear in Section 5.4 - 5.5. The primitives used in our upper bounds and the security model appear in Chapter 2.

5.2 Lower Bounds for (fn|ua)-BoBW

In this section, we show two lower bounds concerning (fn|ua)-BoBW protocols– one with no setup and the other with private setup. In the plain model, we show that it is impossible to design a 4-round (fn|ua)-BoBW protocol (with black-box simulation). In the CRS setting, the 3-round lower bound for (fn|ua)-BoBW protocols follows directly from the impossibility of 2-round protocol achieving fn [108, 102, 166]. However, they do not hold in the presence of PKI. While the argument of [108] crucially relies on the adversary being able to eavesdrop communication between two honest parties (which does not hold in the presence of PKI), the lower bounds of [102, 166] also do not hold if PKI is assumed (as acknowledged / demonstrated in [102, 168]). In the setting with CRS and PKI, we show impossibility of a 2-round protocol. The proof of both our lower bounds relies on the following theorem, which we formally state and prove below.

Theorem 5.1 *An n-party r-round (fn|ua)-BoBW protocol implies a 2-party $(r-1)$-round maliciously-secure oblivious transfer (OT).*

Proof: We prove the theorem for $n = 3$ parties with $t = 1$ and $s = 2$ which can be extended for higher values of n in a natural manner. Let $\mathcal{P} = \{P_1, P_2, P_3\}$ denote the 3 parties and the adversary $\mathcal{A}$ may corrupt at most two parties. As per the hypothesis, we assume that there exists a r-round (fn|ua)-BoBW protocol protocol π_f that can compute the function f defined as $f((m_0, m_1), (c, R_2), R_3) = ((m_c + R_2 + R_3), m_c, m_c)$ which simultaneously achieves fn when $t = 1$ parties are corrupt and ua when $s = 2$ parties are corrupt. At a high-level, we transform the r-round 3-party protocol π_f among $\{P_1, P_2, P_3\}$ into a $(r-1)$-round 2-party OT protocol between a sender P_S with inputs (m_0, m_1) and a receiver P_R with input c. Before describing the transformation, we present the following lemma:

Lemma 5.1 *Protocol π_f must be such that the combined view of $\{P_2, P_3\}$ at the end of Round $(r-1)$ suffices to compute their output.*

Proof: Consider an adversary $\mathcal{A}$ who corrupts only a minority of the parties ($t = 1$). $\mathcal{A}$ controls party P_1 with the following strategy: P_1 behaves honestly in the first $(r-1)$ rounds while he simply remains silent in Round r (last round). Since P_1 receives all the desired communication throughout the protocol, it follows directly from correctness of π_f that $\mathcal{A}$ must be able to compute the output. Since π_f is assumed to be fair for the case of $t = 1$, it must hold that the honest parties P_2 and P_3 must be able to compute the output without any communication from P_1 in Round r. This implies that the combined view of $\{P_2, P_3\}$ at the end of Round $(r-1)$ must suffice to compute the output. □

Our transformation from π_f to a $(r-1)$-round OT protocol π_{OT} between a sender P_S with inputs (m_0, m_1) and a receiver P_R with input c goes as follows. P_S emulates the role of P_1 during π_f while P_R emulates the role of both parties $\{P_2, P_3\}$ during π_f using random inputs R_2, R_3 respectively. In more detail, let $\mathsf{m}^{\mathsf{r}}_{i\to j}$ denote the communication from P_i to P_j in round r of π_f. Then for $\mathsf{r} \in [r-1]$, the interaction in round r of protocol π_{OT} is the following: P_S sends $\mathsf{m}^{\mathsf{r}}_{1\to 2}$ and $\mathsf{m}^{\mathsf{r}}_{1\to 3}$ to P_R while P_R sends $\mathsf{m}^{\mathsf{r}}_{2\to 1}$ and $\mathsf{m}^{\mathsf{r}}_{3\to 1}$ to P_S. P_R computes the output m_c using the combined view of $\{P_2, P_3\}$ at the end of Round $(r-1)$. P_S outputs nothing. Recall that the output of the OT between (P_S, P_R) is $(\perp, m_c)$ respectively. We now argue that π_{OT} realizes the OT functionality.

Lemma 5.2 *Protocol π_{OT} realizes the OT functionality.*

Proof: We first prove that π_{OT} is correct. By Lemma 5.1, it follows that P_R emulating the role of both $\{P_2, P_3\}$ of π_f must be able to compute the correct output m_c by the end of Round $(r-1)$. We now consider the security properties. First, we consider a corrupt P_R (emulating the roles of $\{P_2, P_3\}$ in π_f). Since by assumption, π_f is a protocol that should preserve privacy of P_1's input even in the presence of an adversary corrupting $\{P_2, P_3\}$ ($s = 2$ corruptions), the input m_{1-c} of P_S must remain private against a corrupt P_R. Next, we note that privacy of π_f against a corrupt P_1 ($t = 1$ corruption) guarantees that P_1 does not learn anything beyond the output $(m_c + R_2 + R_3)$ in the protocol π_f which leaks nothing about c. It thus follows that a corrupt P_S in π_{OT} emulating the role of P_1 in π_f will also not be able to learn anything about P_R's input c. More formally, we can construct a simulator for the OT protocol π_{OT} for the cases of corrupt P_R and corrupt P_S by invoking the simulator of π_f for the case of dishonest majority ($s = 2$) and honest majority ($t = 1$) respectively. In each case, it follows from the security of π_f that the simulator of π_f would return a view indistinguishable from the real-world view; directly implying the security of the OT protocol π_{OT}. □

Thus, we can conclude that a $(r-1)$-round 2-party OT protocol π_{OT} can be derived from r-round π_f. This concludes the proof of Theorem 5.1. □

Theorem 5.2 *There exists a function f for which there is no 4-round (resp. 2 round) protocol computing f in the plain model (resp. with CRS and PKI) that simultaneously realises– (1) $\mathcal{F}_{\mathsf{fair}}$ (Figure 2.3) when $t < n/2$ parties are corrupted (2) $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2) when $s < n$ parties are corrupted. In the former setting (plain model), we assume black-box simulation.*

Proof: We start with the proof in the plain model, followed by the proof with CRS and PKI. We assume for contradiction that there exists a 4-round $(\mathsf{fn}|\mathsf{ua})$-BoBW protocol (with black-box simulation) in the plain model. Then, it follows from Theorem 5.1 that there must exist a

3-round 2-party maliciously-secure OT protocol with black-box simulation in the plain model. We point that this OT derived as per the transformation of Theorem 5.1 is a bidirectional OT, where each round consists of messages from both the OT sender and the receiver. Using the round-preserving transformation from bidirectional OT to alternating-message OT (where each round consists of a message from only one of the two parties) [60], we contradict the necessity of 4 rounds for alternating OT in the plain model with black-box simulation [95]. This completes the proof for plain model.

Next, we assume for contradiction that there exists a 2-round $(\mathsf{fn}|\mathsf{ua})$-BoBW MPC protocol in the presence of CRS and PKI. Then, it follows from Theorem 5.1 that there exists 1-round OT protocol in this model. We have arrived at a contradiction since non-interactive OT is impossible to achieve in a model with input-independent setup that includes CRS and PKI (notably 1-round OT constructions which use an *input-dependent* PKI setup such as [25] exist). To be more specific, a 1-round OT protocol would be vulnerable to the following residual attack by a corrupt receiver P_R: P_R can participate in the OT protocol with input c and get the output m_c at the end of the 1-round OT protocol (where (m_0, m_1) denote the inputs of sender P_S). Now, since the Round 1 messages of P_S and P_R are independent of each other, P_R can additionally plug in his input as being $(1-c)$ to locally compute m_{1-c} as well which is a violation of sender's security as per the ideal OT functionality. □

5.3 Upper Bounds for $(\mathsf{fn}|\mathsf{ua})$-BoBW

In this section, we construct two upper bounds for the $(\mathsf{fn}|\mathsf{ua})$-BoBW class. Our upper bounds take 5 and 3 rounds in the plain model and in the CRS setting respectively, tightly matching the lower bounds presented in Section 5.2. We begin with a general compiler that transforms any n-party r-round actively-secure MPC protocol achieving ua in dishonest majority into an $(r+1)$-round $(\mathsf{fn}|\mathsf{ua})$-BoBW protocol.

5.3.1 The Compiler

Drawing motivation from the compiler of [130] from ua to fn in the honest majority setting, our compiler uses the given r-round protocol achieving ua security to compute an "authenticated" secret sharing with a threshold of t of the output y and reconstruct the output y during the $(r+1)^{\text{th}}$ round. The correct reconstruction is guaranteed thanks to unanimity offered by the underlying protocol and the authentication mechanism that makes equivocation of a share hard. Alternatively termed as error-correcting secret sharing (ECSS) [130], the authenticated secret sharing was instantiated with pairwise information-theoretic or one-time MAC as a form of authentication. This, when taken as is in our case, imbibes fairness in the honest majority

setting as in the original transformation. The sharing threshold t ensures that the shares of the honest set, consisting of at least $t+1$ parties, dictate the reconstruction of the output, no matter whether the corrupted parties cooperate or not. The pairwise MAC, however, makes it challenging to maintain unanimity in the dishonest majority case of the transformed protocol, where a corrupt party may choose to verify its share to *selected* few enabling their output reconstruction. This seems to call for a MAC that cannot be manipulated part-wise to keep the verifiers on different pages. A possible approach to achieve the property of public verifiability is by means of digital signatures i.e each party obtains a signed output share which it broadcasts during reconstruction and can be verified by remaining parties using a common public verification key (that the parties obtain as part of the output of the r-round protocol achieving ua). However, we opt for an alternate solution that avoids the use of digital signatures, enabling us to achieve the desirable property of the compiler (transforming any n-party r-round actively-secure MPC protocol achieving ua in dishonest majority into an $(r+1)$-round $(\mathsf{fn}|\mathsf{ua})$-BoBW protocol) being information-theoretic (i.t). Achieving i.t security is a worthwhile goal, as substantiated by its extensive study in numerous settings including those where achieving this desirable security notion demands additional tools. For instance, there are well-known results circumventing the impossibility of achieving i.t security in dishonest majority by relying on additional assistance such as tamper-proof hardware tokens [110, 128, 80] and Physically Uncloneable Functions (PUFs) [164, 45]. Having an i.t compiler opens up the possibility of achieving i.t BoBW MPC by plugging in an i.t. secure dishonest majority protocol (say, that uses hardware tokens / PUFs or some other assistance) in the compiler.

Our i.t compiler is realized via a clean trick inspired from a form of authentication used in the Information Checking Protocol (ICP) primitive of [173, 165] and unanimously identifiable commitments (UIC) of [128]. A value s is authenticated using a 'joint' MAC which is a t-degree (uniform) polynomial $a(x)$ over a field with constant term s. Each verifier P_j receives evaluation of $a(x)$ at a random secret point K_j as verification information– $(\mathsf{K}_j, a(\mathsf{K}_j))$. The secret random points when picked from large enough field make it *statistically* hard for a corrupt authenticator to lie about the MAC polynomial (and the underlying secret) that can cause disagreement across the verifiers. We now define authentication with public verifiability and authenticated t-sharing below. Subsequently, we present a protocol for reconstruction of an authenticated t-shared value and capture the unanimity it offers in a lemma (Lemma 5.3). The protocol and the lemma are used in our compiler and its security proof respectively.

Definition 5.1 (Authentication with Public Verifiability) *A value $s \in \mathbb{F} = GF(2^\kappa)$ is said to be authenticated with public verifiability with an authenticator P and n verifiers $\mathcal{P} =$*

$\{P_1, \ldots, P_n\}$, *if the designated authenticator holds a polynomial* $a(x)$ *of degree at most* t *over* $\mathbb{F}$, *picked uniformly at random, with the constraint that* $a(0) = s$ *and each verifier* P_i *holds* $v_i = (\mathsf{K}_i, a(\mathsf{K}_i))$ *for a random secret value* $\mathsf{K}_i \in \mathbb{F} \setminus \{0\}$. $a(x)$ *is denoted as MAC and* v_i *as the corresponding verification information of verifier* P_i.

Definition 5.2 (Authenticated t-sharing) *A value* $s \in \mathbb{F} = GF(2^\kappa)$ *is said to be authenticated* t*-shared (refer to Section 2.4.3 for* t*-sharing) amongst* n *parties* $\{P_1, \ldots, P_n\}$ *if there exists a polynomial* $p(x)$ *of degree at most* t *over* $\mathbb{F}$, *picked uniformly at random, with the constraint that* $p(0) = s$, *such that each share* $s_i = p(i)$ *of* s *is authenticated with public verifiability w.r.t. authenticator* P_i *and verifiers* $\mathcal{P}$ *and* j^{th} *verifier holding common point* K_j *for all authentication instances. Each* P_i *holds* $a_i(x)$ *as the MAC of* s_i *and* $v_{ij} = (\mathsf{K}_i, a_j(\mathsf{K}_i))$ *as the verification information corresponding to MAC* $a_j(x)$ *held by* P_j.

Protocol Rec

Input: Party P_i holds $\big(a_i(x), \{v_{ij} = (\mathsf{K}_i, a_j(\mathsf{K}_i))\}_{j \in [n]}\big)$.

Output: Secret s or $\perp$

Round 1: P_i broadcasts $a_i(x)$. If $a_j(x)$ broadcasted by P_j is a polynomial of degree at most t and is consistent with v_{ij}, then P_i adds j in a set $\mathcal{V}_i$, marked as verified, which is initialized to $\{i\}$. If $|\mathcal{V}_i| \geq t+1$ and $\{a_j(0)\}_{j \in \mathcal{V}_i}$ lie on a t-degree polynomial, it reconstructs the secret s as the constant term of the interpolated polynomial. Else it outputs $\perp$.

Figure 5.1: Protocol Rec to reconstruct an authenticated t-shared value

Lemma 5.3 *All the honest parties either output* s *or* $\perp$ *in* Rec *(Figure 5.1), except with probability at most* $\frac{n^2}{|\mathbb{F}|-1}$.

Proof: To prove the lemma, we show that the respective $\mathcal{V}$ sets held by all honest parties are identical and do not comprise of *any* j such that P_j broadcasts an incorrect MAC polynomial $a_j^*(x) \neq a_j(x)$, except with probability at most $\frac{n^2}{|\mathbb{F}|-1}$. The latter condition would prove that the reconstructed secret (if any) would be s while the former would show that all honest parties compute the same output. With $\mathbb{F} = GF(2^\kappa)$, the above probability is negligible in κ.

First, consider an honest P_i with verification information $v_{ij} = \big(\mathsf{K}_i, a_j(\mathsf{K}_i)\big)$ corresponding to MAC $a_j(x)$ held by P_j. According to Rec, P_i would include j in $\mathcal{V}_i$ only if $a_j^*(x)$ broadcast by P_j is consistent with v_{ij}. Since a potentially corrupt P_j has no information about the random secret point K_i, the probability that P_j broadcasts $a_j^*(x) \neq a_j(x)$ but $a_j^*(\mathsf{K}_i) = a_j(\mathsf{K}_i)$

is the probability that P_j guessed the secret point K_i correctly which is $\frac{1}{|\mathbb{F}|-1}$ (K_i was picked uniformly at random from $\mathbb{F} \setminus \{0\}$). Extending this argument to all potentially corrupt P_j's, the probability that $\mathcal{V}_i$ includes at least one j such that $a_j^*(x) \neq a_j(x)$ is at most $\frac{|\mathcal{C}|}{|\mathbb{F}|-1}$ (applying union bound), where $\mathcal{C}$ is the set of parties controlled by the adversary $\mathcal{A}$. Finally, applying the union bound over the set of honest parties $\mathcal{H}$, we conclude that the probability that at least one honest party includes some j in its $\mathcal{V}$ set such that P_j broadcast $a_j^*(x) \neq a_j(x)$ is at most $\frac{|\mathcal{H}|\cdot|\mathcal{C}|}{|\mathbb{F}|-1}$. Taking into account that $|\mathcal{H}|, |\mathcal{C}| < n$, this probability is bounded by $\frac{n^2}{|\mathbb{F}|-1}$. Thus all honest parties would have identical $\mathcal{V}$ sets, excluding js such that P_j broadcast the incorrect MAC polynomial, except with probability $\frac{n^2}{|\mathbb{F}|-1}$.

□

We present our protocol $\pi_{\mathsf{bw.fair}}$ in Figure 5.3. The correctness and security of $\pi_{\mathsf{bw.fair}}$ are analyzed in Theorem 5.3 and Theorem 5.4, respectively, in a hybrid-execution model where the parties have access to a functionality $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$ that computes the authenticated t-sharing of the output $y = f(x_1 \dots x_n)$ with ua security. The description of $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$ appears below in Figure 5.2.

Ideal Functionality $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$

Input: On message $(\mathsf{sid}, \mathsf{Input}, x_i, \mathsf{K}_i)$ from a party P_i ($i \in [n]$), do the following: if $(\mathsf{sid}, \mathsf{Input}, *)$ message was received from P_i, then ignore. Otherwise record it internally. If (x_i, K_i) is outside of the domain for P_i ($i \in [n]$), consider $x_i = \texttt{abort}$.

Output to adversary: If there exists $i \in [n]$ such that $x_i = \texttt{abort}$, send $(\mathsf{sid}, \texttt{Output}, \perp)$ to all the parties. Else, compute $y = f(x_1 \dots x_n)$ and compute the authenticated t-sharing of secret $s = y$ (Definition 5.2). Let $p(x)$ denote the t-degree polynomial with $p(0) = y$, $a_i(x)$ denote the MAC of $s_i = p(i)$ and $v_{ij} = \big(\mathsf{K}_i, a_j(\mathsf{K}_i)\big)$ represent the verification information corresponding to MAC $a_j(x)$ held by P_j. Set $z_i = \big(a_i(x), \{v_{ij} = (\mathsf{K}_i, a_j(\mathsf{K}_i))\}_{j\in[n]}\big)$. Send $(\mathsf{sid}, \texttt{Output}, \{z_i\}_{i\in\mathcal{C}})$ to the adversary, where $\mathcal{C}$ denotes the set of parties controlled by the adversary.

Output to honest parties: Receive either `continue` or `abort` from adversary. In case of `continue`, send z_i to each honest P_i, whereas in case of `abort` send $\perp$ to all honest parties.

Figure 5.2: Ideal Functionality $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$

Protocol $\pi_{\mathsf{bw.fair}}$

Inputs: Party P_i has x_i for $i \in [n]$

Model: $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$- hybrid model (Figure 5.2)

Output: $y = f(x_1 \dots x_n)$ or $\perp$

Round $1 - r$**:** P_i interacts with $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$ with input (x_i, K_i) to compute authenticated t-sharing of output $y = f(x_1 \dots x_n)$, where K_i denotes its secret random key from $\mathbb{F} \setminus \{0\}$.

Round $(r+1)$**:** If $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$ returns $\perp$, P_i outputs $\perp$. Else it participates in Rec with the output obtained from $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$ and outputs the output of Rec.

Figure 5.3: ($\mathsf{fn}|\mathsf{ua}$)-BoBW protocol

Theorem 5.3 *Protocol* $\pi_{\mathsf{bw.fair}}$ *is correct, except with negligible probability.*

Proof: We argue that an honest party's output y which is not $\perp$ is correct, with very high probability. In $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$-hybrid model, the output of $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$ is indeed a correct authenticated t-sharing of the output $y = f(x_1 \dots x_n)$ where x_i denotes the input committed by P_i to $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$. In the honest majority setting (i.e. $t < n/2$), $|\mathcal{V}_i|$ of an honest P_i will contain all the honest parties. Therefore, the reconstructed polynomial via the points $\{a_j(0)\}_{j \in \mathcal{V}_i}$ is indeed the correct polynomial and computes the correct output y. In the dishonest majority setting (i.e. $s < n$), $|\mathcal{V}_i|$ of an honest P_i may contain a corrupt party P_j broadcasting a wrong $a_j(x)$ with probability at most $\frac{s}{|\mathbb{F}|-1}$ and as a consequence a wrong t-degree polynomial may get reconstructed. Therefore, except with probability $\frac{s}{|\mathbb{F}|-1}$, P_i's reconstructed output is correct. □

Theorem 5.4 *Protocol* $\pi_{\mathsf{bw.fair}}$ *realises– (i)* $\mathcal{F}_{\mathsf{fair}}$ *(Figure 2.3) when at most* $t < n/2$ *parties are corrupt and (ii)* $\mathcal{F}_{\mathsf{ua}}$ *(Figure 2.2) when at most* $s < n$ *parties are corrupt, in the* $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$*-hybrid model. It takes* $(r+1)$ *rounds, assuming the realization of* $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$ *requires* r *rounds.*

We defer the proof of Theorem 5.4 to Section 5.6.2.

5.3.2 The Upper Bounds

Building our round-optimal ($\mathsf{fn}|\mathsf{ua}$)-BoBW protocols in the plain and CRS model involves constructing 2 and 4 round protocols that achieve ua security against dishonest majority in the respective models. Such protocols when plugged in our compiler of Section 5.3.1 to realise the functionality $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$ would directly yield the round-optimal ($\mathsf{fn}|\mathsf{ua}$)-BoBW protocols. In the CRS setting, the known 2-round protocols of [93, 35] achieve ua and thereby lead to a 3-round ($\mathsf{fn}|\mathsf{ua}$)-BoBW protocol, matching the lower bound. Unfortunately, the existing 4-round MPC protocols in the plain model relying on polynomial-time assumptions [113, 15, 60], in spite of convenient use of broadcast, only satisfy the weaker notion of sa. In this work, we demonstrate how the protocol of [113] and [15, 60] can be tweaked to achieve ua in Appendix 5.7. The former reuses the technique of authentication with public verifiability introduced previously and involves a few other tinkering. With respect to the above-mentioned realizations of

$\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$, our ($\mathsf{fn}|\mathsf{ua}$)-BoBW MPC protocols rely on the assumption of 2-round OT in the common random/reference string model and 4-round OT in the plain model.

Theorem 5.5 *Assuming the existence of a 4 (resp., 2) round MPC protocol that realizes $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2) for upto $n-1$ malicious corruptions in the plain (resp., CRS) model, there exists a 5 (resp., 3)-round MPC protocol in the plain (resp., CRS) model that simultaneously realises– (1) $\mathcal{F}_{\mathsf{fair}}$ (Figure 2.3) when $t < n/2$ parties are corrupted (2) $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2) when $s < n$ parties are corrupted.*

A minor observation regarding the use of broadcast in our compiler is that we can replace it with point-to-point communication at the expense of relaxing ua to sa security in the dishonest majority setting.

Security with Identifiability. Our compiler preserves the property of identifiability. Since the underlying dishonest-majority protocols [93, 35] can be augmented with NIZK in the CRS model to achieve identifiable abort, the upper bound in the CRS model achieves identifiable fairness and abort in the honest and dishonest majority setting respectively. With respect to the plain model, we show how security of [15] can be boosted to achieve identifiable abort with minor tweaks in Appendix 5.7.2. This variant, when compiled using our compiler of Section 5.3.1 would achieve identifiable fairness and abort in the honest and dishonest majority setting respectively. We therefore get a version of Theorem 5.5 where $\mathcal{F}_{\mathsf{ua}}$ and $\mathcal{F}_{\mathsf{fair}}$ are replaced with $\mathcal{F}_{\mathsf{idua}}$ (Figure 2.5) and $\mathcal{F}_{\mathsf{idfair}}$ (Figure 2.6) respectively.

5.4 Lower Bounds for ($\mathsf{god}|\mathsf{ua}$)-BoBW

In this section, we prove that it is impossible to design a 2-round ($\mathsf{god}|\mathsf{ua}$)-BoBW protocol with $t + s < n$ in the CRS model. Note that the necessity of 3 rounds for ($\mathsf{god}|\mathsf{ua}$)-BoBW protocol for most values of (n, s, t) follows from the 2-round impossibility of fair MPC for honest majority in the CRS model [108, 166, 102]. Accounting for the fact that these existing results do not rule out the possibility of 2-round ($\mathsf{god}|\mathsf{ua}$)-BoBW MPC for $(t = 1, s > t, n \geq 4)$, we present a *unified proof* that works even for $s > t$, for all values of t (including $t = 1$). Our proof approach deals with adversarial strategies that shuffle between the honest and dishonest majority setting, highlighting the challenge of designing protocols that simultaneously provide different guarantees for different settings. This is in contrast to the existing lower bounds of [108, 166, 102] which deal only with honest majority setting and single security notion of fn. Lastly, we demonstrate why our proof breaks down in the presence of PKI. Indeed, we construct a 2-round ($\mathsf{god}|\mathsf{ua}$)-BoBW protocol assuming CRS and PKI in this work.

Theorem 5.6 *Let n, t, s be such that $t + s < n$ and $t < n/2$. There exist functions f for which there is no two-round protocol in the CRS model computing f that simultaneously realizes– (1) $\mathcal{F}_{\mathsf{god}}$ (Figure 2.4) when $t < n/2$ parties are corrupted (2) $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2) when $s < n$ parties are corrupted.*

Proof: We prove the theorem for $n = 4$ parties with $t = 1$ and $s = 2$. The result then can be extended for higher values of n in a natural manner. Let $\mathcal{P} = \{P_1, P_2, P_3, P_4\}$ denote the set of 4 parties and $\mathcal{A}$ may corrupt at most two among them. We prove the theorem by contradiction. We assume that there exists a 2-round ($\mathsf{god}|\mathsf{ua}$) BoBW protocol π in the CRS model that can compute the function $f(x_1, x_2, x_3, x_4)$ defined below for P_i's input x_i: $f(x_1, x_2, x_3, x_4) = 1$ if $x_1 = x_2 = 1; 0$ otherwise. By assumption, π achieves god when $t = 1$ parties are corrupt and ua security when $s = 2$ parties are corrupt (satisfying feasibility criteria $t + s < n$).

At a high level, we discuss three adversarial strategies $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}_3$ of $\mathcal{A}$. While both $\mathcal{A}_1$ and $\mathcal{A}_3$ deal with $t = 1$ corruption with the adversary corrupting P_1, $\mathcal{A}_2$ involves $s = 2$ corruptions where the adversary corrupts $\{P_3, P_4\}$. We consider $\mathcal{A}_i$ strategy as being launched in execution Σ_i ($i \in [3]$) of π. The executions are assumed to be run for the same input tuple $(x_1, x_2, \perp, \perp)$ and the same random inputs (r_1, r_2, r_3, r_4) of the parties. (Same random inputs are considered for simplicity and without loss of generality. The same arguments hold for distribution ensembles as well.) Our executions and adversarial strategies are sequenced in the order of increasingly more non-cooperating malicious adversaries. Yet, keeping the views of a certain party between two consecutive executions same, we are able to conclude the party would output the correct value even in the face of stronger malicious behaviour. Finally, we reach to the final execution Σ_3 where we show that a party can deduce the output in the end of Round 1 itself. Lastly, we show a strategy for the party to explicitly breach the input privacy of one of the input-contributing parties.

We assume that the communication done in the second round of π is via broadcast alone. This holds without loss of generality since the parties can perform point-to-point communication by exchanging random pads in the first round and then use these random pads to unmask later broadcasts. We use the following notation: Let $\mathsf{p}^1_{i \to j}$ denote the pairwise communication from P_i to P_j in round 1 and b^r_i denote the broadcast by P_i in round r, where $r \in [2], \{i, j\} \in [4]$. These values may be function of CRS as per the working of the protocol. V^ℓ_i denotes the view of party P_i at the end of execution Σ_ℓ ($\ell \in [3]$) of π. Below we describe the strategies $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}_3$.

$\mathcal{A}_1$: $\mathcal{A}$ corrupts P_1 here. P_1 behaves honestly towards P_2 in Round 1, i.e sends the messages

$\mathsf{p}^1_{1\to 2}, \mathsf{b}^1_1$ as per the protocol. However P_1 does not communicate privately to $\{P_3, P_4\}$ in Round 1. In Round 2, P_1 behaves honestly as per the protocol.

$\mathcal{A}_2$**:** $\mathcal{A}$ corrupts $\{P_3, P_4\}$ here. $\{P_3, P_4\}$ behave honestly in Round 1 of the protocol. In Round 2, P_k ($k \in \{3, 4\}$) acts as per the protocol specification when no *private* message from P_1 is received in Round 1. Specifically, suppose P_k did not receive $\mathsf{p}^1_{1\to k}$ in Round 1. Let $\overline{\mathsf{b}^2_k}$ denote the message that should be sent by P_k as per the protocol in Round 2 in such a scenario. Then as per $\mathcal{A}_2$, corrupt P_k sends $\overline{\mathsf{b}^2_k}$ in Round 2.

$\mathcal{A}_3$**:** Same as in $\mathcal{A}_1$ and in addition– during Round 2, P_1 simply remains silent i.e waits to receive the messages from other parties, but does not communicate at all.

Next we present the views of the parties in Σ_1, Σ_2 and Σ_3 in Table 5.2. Here, $\overline{\mathsf{b}^2_k}$ ($k \in \{3, 4\}$) denotes the message that should be sent by P_k according to the protocol in Round 2 in case P_k did not receive any private communication from P_1 in Round 1.

	Σ_1				Σ_2				Σ_3			
	V^1_1	V^1_2	V^1_3	V^1_4	V^2_1	V^2_2	V^2_3	V^2_4	V^3_1	V^3_2	V^3_3	V^3_4
Input	(x_1, r_1)	(x_2, r_2)	r_3	r_4	(x_1, r_1)	(x_2, r_2)	r_3	r_4	(x_1, r_1)	(x_2, r_2)	r_3	r_4
R1	$\mathsf{p}^1_{2\to 1}, \mathsf{p}^1_{3\to 1}$ $\mathsf{p}^1_{4\to 1}$, $\mathsf{b}^1_2, \mathsf{b}^1_3, \mathsf{b}^1_4$	$\mathsf{p}^1_{1\to 2}, \mathsf{p}^1_{3\to 2}$, $\mathsf{p}^1_{4\to 2}$, $\mathsf{b}^1_1, \mathsf{b}^1_3, \mathsf{b}^1_4$	$-, \mathsf{p}^1_{2\to 3}$, $\mathsf{p}^1_{4\to 3}$, $\mathsf{b}^1_1, \mathsf{b}^1_2, \mathsf{b}^1_4$	$-, \mathsf{p}^1_{2\to 4}$, $\mathsf{p}^1_{3\to 4}$, $\mathsf{b}^1_1, \mathsf{b}^1_2, \mathsf{b}^1_3$	$\mathsf{p}^1_{2\to 1}, \mathsf{p}^1_{3\to 1}$ $\mathsf{p}^1_{4\to 1}$, $\mathsf{b}^1_2, \mathsf{b}^1_3, \mathsf{b}^1_4$	$\mathsf{p}^1_{1\to 2}, \mathsf{p}^1_{3\to 2}$, $\mathsf{p}^1_{4\to 2}$, $\mathsf{b}^1_1, \mathsf{b}^1_3, \mathsf{b}^1_4$	$\mathsf{p}^1_{1\to 3}, \mathsf{p}^1_{2\to 3}$, $\mathsf{p}^1_{4\to 3}$, $\mathsf{b}^1_1, \mathsf{b}^1_2, \mathsf{b}^1_4$	$\mathsf{p}^1_{1\to 4}, \mathsf{p}^1_{2\to 4}$, $\mathsf{p}^1_{3\to 4}$, $\mathsf{b}^1_1, \mathsf{b}^1_2, \mathsf{b}^1_3$	$\mathsf{p}^1_{2\to 1}, \mathsf{p}^1_{3\to 1}$ $\mathsf{p}^1_{4\to 1}$, $\mathsf{b}^1_2, \mathsf{b}^1_3, \mathsf{b}^1_4$	$\mathsf{p}^1_{1\to 2}, \mathsf{p}^1_{3\to 2}$, $\mathsf{p}^1_{4\to 2}$, $\mathsf{b}^1_1, \mathsf{b}^1_3, \mathsf{b}^1_4$	$-, \mathsf{p}^1_{2\to 3}$, $\mathsf{p}^1_{4\to 3}$, $\mathsf{b}^1_1, \mathsf{b}^1_2, \mathsf{b}^1_4$	$-, \mathsf{p}^1_{2\to 4}$, $\mathsf{p}^1_{3\to 4}$, $\mathsf{b}^1_1, \mathsf{b}^1_2, \mathsf{b}^1_3$
R2	$\mathsf{b}^2_2, \overline{\mathsf{b}^2_3}, \overline{\mathsf{b}^2_4}$	$\mathsf{b}^2_1, \overline{\mathsf{b}^2_3}, \overline{\mathsf{b}^2_4}$	$\mathsf{b}^2_1, \mathsf{b}^2_2, \overline{\mathsf{b}^2_4}$	$\mathsf{b}^2_1, \mathsf{b}^2_2, \overline{\mathsf{b}^2_3}$	$\mathsf{b}^2_2, \overline{\mathsf{b}^2_3}, \overline{\mathsf{b}^2_4}$	$\mathsf{b}^2_1, \overline{\mathsf{b}^2_3}, \overline{\mathsf{b}^2_4}$	$\mathsf{b}^2_1, \mathsf{b}^2_2, \overline{\mathsf{b}^2_4}$	$\mathsf{b}^2_1, \mathsf{b}^2_2, \overline{\mathsf{b}^2_3}$	$\mathsf{b}^2_2, \overline{\mathsf{b}^2_3}, \overline{\mathsf{b}^2_4}$	$-, \overline{\mathsf{b}^2_3}, \overline{\mathsf{b}^2_4}$	$-, \mathsf{b}^2_2, \overline{\mathsf{b}^2_4}$	$-, \mathsf{b}^2_2\ \overline{\mathsf{b}^2_3}$

Table 5.2: Views of P_1, P_2, P_3, P_4 in Σ_1, Σ_2, Σ_3

We now prove a sequence of lemmas to complete our proof. Let y denote the output computed as per the inputs (x_1, x_2) provided by the honest P_1 and P_2.

Lemma 5.4 *The view of P_2 is the same in Σ_1 and Σ_2 and it outputs y in both.*

Proof: We observe that as per both strategies $\mathcal{A}_1$ and $\mathcal{A}_2$, P_2 receives communication from P_1, P_3, P_4 as per honest execution in Round 1. In Round 2, according to $\mathcal{A}_1$, corrupt P_1 did not send private messages to P_3 and P_4 who therefore broadcast $\overline{\mathsf{b}^2_3}$ and $\overline{\mathsf{b}^2_4}$ respectively as per protocol specification. On the other hand, according to $\mathcal{A}_2$, corrupt P_3 and corrupt P_4 send the same messages respectively as per protocol specification for case when P_3, P_4 receive no private message from P_1 in Round 1. It is now easy to check (refer Table 5.2) that $\mathsf{V}^1_2 = \mathsf{V}^2_2$. Now, since Σ_1 involves $t = 1$ corruption, by assumption, π must be robust and V^1_2 must lead to output computation, say of output y'. Due to view equality, P_2 in Σ_2 must also output y'. In

Σ_2, P_1 and P_2 are honest and their inputs are x_1 and x_2 respectively. Due to correctness of π during Σ_2, it must then hold that $y' = y$ i.e the output computed based on V_2^2 is according to honest P_1's input x_1 during Σ_2. This completes the proof. □

Lemma 5.5 *The view of P_1 is the same in Σ_2 and Σ_3 and it outputs y in both.*

Proof: An honest P_2 has the same view in both Σ_1 and Σ_2 and outputs y as per Lemma 5.4. As π achieves ua in the presence of $s = 2$ corruptions, P_1 learns y in Σ_2. We now show that P_1's view in Σ_2 and Σ_3 are the same and so it outputs y in Σ_3. First, it is easy to see that the Round 1 communication towards P_1 is as per honest execution in both Σ_2, Σ_3. Next, recall that as per $\mathcal{A}_2$, both corrupt $\{P_3, P_4\}$ send messages in Round 2 according to the scenario when they didn't receive any private communication from P_1 in Round 1. A similar message would be sent by honest $\{P_3, P_4\}$ in Σ_3 who did not receive private message from corrupt P_1 as per $\mathcal{A}_3$. Finally, since corrupt P_1 behaved honestly to P_2 in Round 1 of Σ_3 as per $\mathcal{A}_3$, the Round 2 communication from P_2 is similar to that in execution Σ_2. It is now easy to verify (refer Table 5.2) that $\mathsf{V}_1^2 = \mathsf{V}_1^3$ from which output y can be computed. □

Lemma 5.6 *P_2 in Σ_3 should learn the output y by the end of Round 1.*

Proof: Firstly, it follows directly from Lemma 5.5 and the assumption that protocol π is robust against $t = 1$ corruption that all parties including P_2 must learn output y at the end of Σ_3. Next, we note that as per strategy $\mathcal{A}_3$, P_1 only communicates to P_2 in Round 1. We argue that the second round communication from P_3, P_4 does not impact P_2's output computation as follows: we observe that the output y depends only on (x_1, x_2). Clearly, Round 1 messages of P_3, P_4 does not depend on x_1. Next, since there is no private communication to P_3, P_4 from P_1 as per strategy $\mathcal{A}_3$, the only communication that can possibly hold information on x_1 and can impact the round 2 messages of P_3, P_4 is b_1^1. However, since this is a broadcast message, P_2 also holds this by the end of Round 1 itself. Thus, P_2 must be able to compute the output y at the end of Round 1.

In more detail, P_2 can choose randomness r_3, r_4 on behalf of P_3, P_4 to locally emulate their following Round 1 messages $\{\mathsf{p}_{3\to2}^1, \mathsf{p}_{4\to2}^1, \mathsf{p}_{3\to4}^1, \mathsf{p}_{4\to3}^1, \mathsf{b}_3^1, \mathsf{b}_4^1\}$. Next, P_2 can now simulate P_3's Round 2 message $\overline{\mathsf{b}_3^2}$ which is a function of its view comprising of $\{\mathsf{p}_{2\to3}^1, \mathsf{p}_{4\to3}^1, \mathsf{b}_1^1, \mathsf{b}_2^1, \mathsf{b}_4^1\}$ (all of which are available to P_2, where b_1^1 was broadcast by P_1 in Round 1). Similarly, P_2 can locally compute P_4's Round 2 message $\overline{\mathsf{b}_4^2}$. We can thus conclude that P_2's view at the end of Σ_3 comprising of $\{\mathsf{p}_{1\to2}^1, \mathsf{p}_{3\to2}^1, \mathsf{p}_{4\to2}^1, \mathsf{b}_1^1, \mathsf{b}_3^1, \mathsf{b}_4^1, \overline{\mathsf{b}_3^2}, \overline{\mathsf{b}_4^2}\}$ can be locally simulated by him at the end of Round 1 itself from which the output y can be computed. □

Lemma 5.7 *A corrupt P_2 violates the privacy property of π.*

Proof: The adversary corrupting P_2 participates in the protocol honestly by fixing input $x_2 = 0$. Since P_2 can get the output at the end of Round 1 (Lemma 5.6), it must be true that P_2 can evaluate f locally by plugging in any value of x_2. Now a corrupt P_2 can plug in $x_2 = 1$ locally and learn x_1 (via the output $x_1 \wedge x_2$). In the ideal world, corrupt P_2 must learn nothing beyond the output 0 as it has participated in the protocol with input 0. But in the execution of π (in which P_2 participated honestly with input $x_2 = 0$), P_2 has learnt x_1. This is a breach of privacy as P_2 learns x_1 regardless of his input. □

Hence, we have arrived at a contradiction, completing proof of Theorem 5.6. □

We draw attention to the fact that Lemma 5.6 would not hold in the presence of any additional setup such as PKI. With additional setup, P_3, P_4 may possibly hold some private information (such as their secret key in case of PKI used to decode P_1's broadcast message in Round 1) that is not available to P_2. Due to this reason, we cannot claim that P_2 can emulate Round 2 messages of $\{P_3, P_4\}$ locally at the end of Round 1. However, this holds in case of CRS as the knowledge of CRS is available to all parties at the beginning of the protocol.

5.5 Upper Bounds for (god|ua)-BoBW

In this section, we present three (god|ua)-BoBW MPC protocols, assuming $t + s < n$ which is the feasibility condition for such protocols ([127]) consuming– *a*) 3-rounds with CRS *b*) 2-rounds with an additional PKI setup *c*) 5-rounds in plain model. The first two are round-optimal in light of the lower bound of Section 5.4 and [112] respectively. The third construction is nearly round-optimal (falls just one short of the 4-round lower bound of [95]). Among our upper bounds, the construction in the plain model is considerably more involved and uses several new tricks in conjugation with existing techniques.

5.5.1 (god|ua)-BoBW MPC with Public and Private Setup

To arrive at the final destination, the roadmap followed is: (i) A 2-round MPC achieving ua security is compiled to a 3-round (god|ua)-BoBW MPC protocol, both against a weaker semi-malicious adversary. With the additional provision of PKI, this compiler can be turned to a round-preserving one. (ii) The semi-malicious (god|ua)-BoBW MPC protocols are compiled to malicious ones in CRS setting via the known round-preserving compiler of [10] (using NIZKs). All the involved and resultant constructions are in *broadcast-only* setting. The protocol just with CRS tightly upper bounds the 3-round lower bound presented in Section 5.4, which accounts for both pair-wise and broadcast channels. The protocol with additional PKI setup works in 2 rounds, displaying the power of PKI and that our lower bound of 3-rounds in Theorem 5.6 breaks down in the presence of PKI. Yet, this construction is round optimal, in light of the

known impossibility of 1-round MPC [112].

5.5.1.1 3-round (god|ua)-BoBW MPC in semi-malicious setting.

In this section, we present a generic compiler that transforms any 2-round MPC protocol $\pi_{\mathsf{ua.sm}}$ achieving ua security into a 3-round broadcast-only (god|ua)-BoBW MPC protocol $\pi_{\mathsf{bw.god.sm}}$ assuming $t+s<n$. Our compiler borrows techniques from the compiler of [4] which is designed for the honest majority setting and makes suitable modifications to obtain BoBW guarantees. Recall that a semi-malicious adversary needs to follow the protocol specification, but has the liberty to decide the input and random coins in each round. Additionally, the parties controlled by the semi-malicious adversary may choose to abort at any step. For completeness, semi-malicious security is defined in Section 5.6.1. The underlying and the resultant protocol use broadcast as the *only* medium of communication.

To transform $\pi_{\mathsf{ua.sm}}$ to guarantee BoBW security, the compiler banks on the idea of giving out the Round 2 message of $\pi_{\mathsf{ua.sm}}$ in a way that ensures god in case of honest majority. The dishonest majority protocols usually do not provide this feature even against a single corruption, let alone a minority. Mimicking the Round 1 of $\pi_{\mathsf{ua.sm}}$ as is, $\pi_{\mathsf{bw.god.sm}}$ achieves this property by essentially giving out a secret sharing of the Round 2 messages of $\pi_{\mathsf{ua.sm}}$ with a threshold of s. When at most t parties are corrupt, the set of $s+1$ honest parties pool their shares to reconstruct Round 2 messages of $\pi_{\mathsf{ua.sm}}$ and compute the output robustly as in $\pi_{\mathsf{ua.sm}}$. This idea is enabled by encoding (i.e garbling) the next message functions of the second round of $\pi_{\mathsf{ua.sm}}$ and secret-sharing their encoding information using a threshold of s in Round 2 and reconstructing the appropriate input labels in the subsequent round. (Refer Section 2.4.1 for details on Garbling Schemes.) The next-message circuit of a party P_i hard-codes Round 1 broadcasts of $\pi_{\mathsf{ua.sm}}$, P_i's input and randomness and the default input and randomness of all the other parties. It takes n flags as input, the j^{th} one indicating the alive/non-alive status of P_j. P_j turning non-alive (aborting) translates to the j^{th} flag becoming 0 in which case the circuit makes sure P_j's default input is taken for consideration by internally recomputing P_j's first round broadcast and subsequently using that to compute the Round 2 message of P_i. Since the flag bits become public by the end of Round 2 (apparent as broadcast is the only mode of communication), the parties help each other by reconstructing the correct label, enabling all to compute the garbled next-message functions of all the parties and subsequently run the output computation of $\pi_{\mathsf{ua.sm}}$. The agreement of the flag bits further ensures output computation is done on a unique set of inputs. The transfer of the shares in broadcast-only setting is enabled via setting up a (public key, secret key) pair in the first round by every party. Broadcasting the encrypted shares emulates sending the share privately. This technique of garbled circuits

computing the augmented next-message function (taking the list of alive (non-aborting) parties as input) followed by reconstruction of the appropriate input label was used in the work of [4] for the honest majority setting. The primary difference in our compiler is with respect to the threshold of the secret-sharing of the labels, to ensure BoBW guarantees.

The formal description of protocol $\pi_{\mathsf{bw.god.sm}}$ appears in Figure 5.4. We analyze its correctness and security below.

Protocol $\pi_{\mathsf{bw.god.sm}}$

Inputs: Party P_i has input x_i and randomness r_i for $i \in [n]$.

Common Inputs: 2- round semi-malicious protocol $\pi_{\mathsf{ua.sm}}$ in the broadcast-only model with $\mathcal{B}_i^\ell$ denoting the message broadcast by P_i in Round ℓ ($\ell \in [2]$). The messages of $\pi_{\mathsf{ua.sm}}$ can be expressed as $\mathcal{B}_i^1 \leftarrow \pi^1_{\mathsf{ua.sm},i}(x_i, r_i)$ and $\mathcal{B}_i^2 \leftarrow \pi^2_{\mathsf{ua.sm},i}(x_i, r_i, T^1)$, where T^1 denotes the transcript of Round 1, namely $(\mathcal{B}_1^1, \ldots \mathcal{B}_n^1)$ and $\pi^1_{\mathsf{ua.sm},i}, \pi^2_{\mathsf{ua.sm},i}$ denote the next-message function for Round 1 and Round 2 respectively of P_i in $\pi_{\mathsf{ua.sm}}$. Finally, let transcript T^2 at the end of Round 2 be defined as $(\{\mathcal{B}_i^1, \mathcal{B}_i^2\}_{i\in[n]})$ and the output computation function of P_i is denoted as $y = \pi^o_{\mathsf{ua.sm},i}(x_i, r_i, T^2)$.

Primitives: Adaptive Garbling Scheme $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ (Section 2.4.1.1) which is projective (assume side-information $\theta(C)$ leaks topology of C), Public-key encryption Scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$

Round 1: Each party P_i initializes $\mathsf{flag}_j = 1$, $\forall j \in [n]$, computes $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ and $\mathcal{B}_i^1 \leftarrow \pi^1_{\mathsf{ua.sm},i}(x_i, r_i)$ and broadcasts $(\mathsf{pk}_i, \mathcal{B}_i^1)$. Let $T^1 = \{\mathcal{B}_1^1, \ldots \mathcal{B}_n^1\}$.

Round 2: Let $C_i(\mathsf{flag}_1, \ldots \mathsf{flag}_n)$ be a circuit that has (x_i, r_i, T^1) and default input and randomness of all parties hardcoded and takes as input n bits $\{\mathsf{flag}_j\}_{j\in[n]}$. C_i acts as follows:

- if $\mathsf{flag}_j = 0$, then recompute $\mathcal{B}_j^1$ in T^1 as per $\pi^1_{\mathsf{ua.sm},j}$ based on default input randomness of P_j, for $j \in [n]$;
- compute $\mathcal{B}_i^2 \leftarrow \pi^2_{\mathsf{ua.sm},i}(x_i, r_i, T^1)$ and output $\mathcal{B}_i^2$.

P_i does the following:

- Run $(\mathsf{C}_i, e_i, d_i) \leftarrow \mathsf{Gb}(1^\lambda, C_i)$ and broadcast (C_i, d_i).
- Let $\{\mathsf{lab}_i^{k,b}\}_{k\in[n],b\in\{0,1\}}$ denote the set of input labels as per e_i. Compute s-sharing of $\mathsf{lab}_i^{k,b}$ for all $k \in [n]$ and $b \in \{0,1\}$ and broadcast $c_{i,j}^{k,b} = \mathsf{Enc}(\mathsf{pk}_j, \mathsf{lab}_{i,j}^{k,b})$ where $\mathsf{lab}_{i,j}^{k,b}$ denotes P_j's share of $\mathsf{lab}_i^{k,b}$. For all $j \in [n]$, $b \in \{0,1\}, k \in [n]$, compute $\mathsf{lab}_{j,i}^{k,b} \leftarrow \mathsf{Dec}(\mathsf{sk}_i, c_{j,i}^{k,b})$.
- Set $\mathsf{flag}_j = 0$ if P_j ($j \in [n]$) aborts in Round 1 or Round 2. If $\mathsf{flag}_j = 0$, then recompute $\mathcal{B}_j^1$ in T^1 based on default input and randomness of P_j.

Round 3: For each C_j obtained in Round 2, P_i participates in the reconstruction of $\{\mathsf{lab}_j^{k,\mathsf{flag}_k}\}_{k\in[n]}$ by broadcasting share $\{\mathsf{lab}_{j,i}^{k,\mathsf{flag}_k}\}_{k\in[n]}$.

Output Computation: Each P_i does the following:

- For each (C_j, d_j) received in Round 2, reconstruct the input labels $\{\mathsf{lab}_j^{k,\mathsf{flag}_k}\}_{k\in[n]}$ using the shares broadcast in Round 3. Output $\perp$ if any reconstruction fails. Else, compute $\mathcal{B}_j^2 \leftarrow \mathsf{De}(\mathsf{Ev}(\mathsf{C}_j, \{\mathsf{lab}_j^{k,\mathsf{flag}_k}\}_{k\in[n]}), d_j)$.
- Corresponding to P_j where $\mathsf{flag}_j = 0$, compute $\mathcal{B}_j^2 \leftarrow \pi^2_{\mathsf{ua.sm},j}(x_j, r_j, T^1)$ using default (x_j, r_j).
- Finally, compute and output $y = \pi^o_{\mathsf{ua.sm},i}(x_i, r_i, T^2)$ with $T^2 = (\{\mathcal{B}_i^1, \mathcal{B}_i^2\}_{i\in[n]})$.

Figure 5.4: 3-round semi-malicious ($\mathsf{god}|\mathsf{ua}$)-BoBW MPC protocol $\pi_{\mathsf{bw.god.sm}}$ from 2-round semi-malicious MPC $\pi_{\mathsf{ua.sm}}$

Theorem 5.7 *Protocol $\pi_{\mathsf{bw.god.sm}}$ is correct, except with negligible probability.*

Proof: We claim that if an honest party outputs $y \neq \perp$, y must be the correct output on the 'committed' inputs of parties. Here, 'committed' refers to the actual inputs for honest parties, inputs written on witness tape at the end of Round 2 for the semi-malicious alive parties and default input for the non-alive parties (who abort in either Round 1 or 2). We first argue that if the reconstruction of an input label is successful, it must correspond to the appropriate public value of flag. This is evident in the honest majority case, as the $(s+1)$ shares contributed by the honest parties would ensure that the reconstruction of the s-shared input label is correct. In the dishonest majority case, we argue that the share (if any) sent by semi-malicious P_j in Round 3 for reconstruction must indeed correspond to the original message (share) encrypted in the ciphertext broadcast in Round 2 using pk_j. This follows from the correctness of the public-key encryption scheme as the semi-malicious P_j will not be able to justify an incorrect share as being a valid decryption of the ciphertext, except with negligible probability. It is now easy to check that the correctness of the adaptive garbling scheme ensures that the garbled circuit evaluated on the appropriate public values of flag would yield the Round 2 message based on the 'committed' inputs; leading to each honest party computing T^2 accordingly. Finally, it follows directly from the correctness of the underlying protocol $\pi_{\mathsf{ua.sm}}$ that the output computed using T^2 by each honest party must be correct. □

Theorem 5.8 *Let (n, s, t) be such that $s + t < n$. Let $\pi_{\mathsf{ua.sm}}$ realises $\mathcal{F}_{\mathsf{ua}}$ for upto $n-1$ semi-malicious corruptions. Then protocol $\pi_{\mathsf{bw.god.sm}}$ realises– (i) $\mathcal{F}_{\mathsf{god}}$ (Figure 2.4) when at most*

$t < n/2$ parties are corrupt and (ii) $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2) when at most $s < n$ parties are corrupt, semi-maliciously in both cases. It takes 3 rounds, assuming that $\pi_{\mathsf{ua.sm}}$ takes 2 rounds.

We defer the proof of Theorem 5.8 to Section 5.6.3.

5.5.1.2 2-round (god|ua)-BoBW MPC in semi-malicious setting

The compiler of the previous section can be made round preserving by assuming pair-wise channels or alternately, PKI. The main difference lies in preponing the actions of Round 2 of $\pi_{\mathsf{bw.god.sm}}$ to Round 1, by exploiting the presence of private channels or PKI.

2-round semi-malicious (god|ua)-BoBW MPC using both broadcast and pairwise-private channels. We observe that the compiler of Section 5.5.1.1 can be modified such that it transforms the 2-round broadcast-only semi-malicious protocol $\pi_{\mathsf{ua.sm}}$ (achieving security with ua) into a 2-round semi-malicious (god|ua)-BoBW MPC protocol $\phi_{\mathsf{bw.god.sm}}$ using both point-to-point and broadcast channel. The 2-round protocol $\phi_{\mathsf{bw.god.sm}}$ is similar to the 3-round broadcast-only protocol $\pi_{\mathsf{bw.god.sm}}$ (Figure 5.4), except for the following differences: The actions of Round 1 and Round 2 of $\pi_{\mathsf{bw.god.sm}}$ are carried out in Round 1 of $\phi_{\mathsf{bw.god.sm}}$. In more detail, Round 1 of $\phi_{\mathsf{bw.god.sm}}$ proceeds as follows - In addition to sending the Round 1 message as per $\pi_{\mathsf{ua.sm}}$, the parties also prepare and send the adaptive garbled circuits meant to compute their Round 2 message of $\pi_{\mathsf{ua.sm}}$ in Round 1 itself. Since the next-message function computing the Round 2 message takes as input the transcript of Round 1, this garbled circuit (being sent in Round 1) will need to take additionally as input the transcript of Round 1 apart from the list of alive (non-aborting) parties (unlike $\pi_{\mathsf{bw.god.sm}}$ where the garbled circuit was sent in Round 2 and thereby only needed to take the list of alive parties as input). Each party s-shares all the input labels of its garbled circuit in Round 1. This step would involve using point-to-point channels to communicate the shares (unlike $\pi_{\mathsf{bw.god.sm}}$ where it was done via broadcast channels in Round 2). Next, in Round 2 of $\phi_{\mathsf{bw.god.sm}}$, similar to Round 3 of $\pi_{\mathsf{bw.god.sm}}$, the reconstruction of the appropriate input labels occur. Note that this can be done as all the values of input wires of the garbled circuit, including the set of alive parties and the transcript of Round 1 are public ($\pi_{\mathsf{ua.sm}}$ is a broadcast-only protocol). This completes the description of $\phi_{\mathsf{bw.god.sm}}$ and it is easy to check that its security can be proved similar to the security of $\pi_{\mathsf{bw.god.sm}}$. This construction is based on [4]. Instantiating $\pi_{\mathsf{ua.sm}}$ with the 2-round broadcast-only semi-malicious protocol of [93, 35], the compiler described above would yield a 2-round (god|ua)-BoBW protocol $\phi_{\mathsf{bw.god.sm}}$ in the semi-malicious setting using both pairwise-private and broadcast channels.

2-round semi-malicious (god|ua)-BoBW MPC using PKI. In the presence of PKI, the protocol $\phi_{\mathsf{bw.god.sm}}$ can be easily transformed to a broadcast-only protocol $\psi_{\mathsf{bw.god.sm}}$. Elaborating

on this, the private messages in $\phi_{\mathsf{bw.god.sm}}$ via the pairwise channel can be emulated in $\psi_{\mathsf{bw.god.sm}}$ by broadcasting the encryption of the private message with the public-key of the intended recipient. This leads to a 2-round broadcast-only (god|ua)-BoBW MPC $\psi_{\mathsf{bw.god.sm}}$ in semi-malicious setting assuming PKI. Both protocols $\phi_{\mathsf{bw.god.sm}}$ and $\psi_{\mathsf{bw.god.sm}}$ are tight upper bounds, in light of the known impossibility of 1-round MPC protocols for any meaningful security notion ([112]).

We state the formal theorems below whose proofs follow similar to proof of Theorem 5.8.

Theorem 5.9 *Let (n, s, t) be such that $s + t < n$. Let $\pi_{\mathsf{ua.sm}}$ realises $\mathcal{F}_{\mathsf{ua}}$ for upto $n - 1$ semi-malicious corruption. Then there exists a protocol $\phi_{\mathsf{bw.god.sm}}$ that uses both broadcast and pairwise-private channel which realises– (i) $\mathcal{F}_{\mathsf{god}}$ (Figure 2.4) when at most $t < n/2$ parties are corrupt and (ii) $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2) when at most $s < n$ parties are corrupt, semi-maliciously in both cases. It takes 2 rounds, assuming that $\pi_{\mathsf{ua.sm}}$ takes 2 rounds.*

Theorem 5.10 *Let (n, s, t) be such that $s + t < n$. Let $\pi_{\mathsf{ua.sm}}$ realises $\mathcal{F}_{\mathsf{ua}}$ for upto $n - 1$ semi-malicious corruption. Then there exists a protocol $\psi_{\mathsf{bw.god.sm}}$, assuming PKI which realises– (i) $\mathcal{F}_{\mathsf{god}}$ (Figure 2.4) when at most $t < n/2$ parties are corrupt and (ii) $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2) when at most $s < n$ parties are corrupt, semi-maliciously in both cases. It takes 2 rounds, given that $\pi_{\mathsf{ua.sm}}$ takes 2 rounds.*

5.5.1.3 The upper bounds with public and private setup

The 2-round semi-malicious broadcast-only protocol of [93, 35] can be plugged in as $\pi_{\mathsf{ua.sm}}$ in our compilers from previous sections to directly yield a 3-round broadcast-only protocol $\pi_{\mathsf{bw.god.sm}}$, 2-round protocol $\phi_{\mathsf{bw.god.sm}}$ that uses both broadcast and pairwise-private channels and 2-round broadcast-only protocol $\psi_{\mathsf{bw.god.sm}}$ assuming PKI, all in the semi-malicious setting. Next, the compiler of [10] that upgrades any broadcast-only semi-malicious protocol to maliciously-secure by employing NIZKs, can be applied on $\pi_{\mathsf{bw.god.sm}}$ and $\psi_{\mathsf{bw.god.sm}}$ to yield a 3-round (god|ua)-BoBW protocol in the CRS model and a 2-round (god|ua)-BoBW protocol given both CRS and PKI. At a high-level, to ensure that the malicious parties indeed follow the description of the protocol, as per the compiler of [10], each party has to prove in zero-knowledge that the message it has produced is consistent with the transcript of the protocol so far. In our compiled protocols, if the zero-knowledge proof of a malicious party, say P_i, fails in a particular Round ℓ; then its message in Round ℓ is interpreted as $\perp$. This scenario is analogous to semi-malicious P_i aborting in the underlying semi-malicious protocol $\pi_{\mathsf{bw.god.sm}}$ during Round ℓ. The BoBW guarantees of the maliciously-secure compiled protocol thereby follow directly from the BoBW guarantees of $\pi_{\mathsf{bw.god.sm}}$ (as $\pi_{\mathsf{bw.god.sm}}$ achieves GOD even if upto t parties abort). However,

while this works to compile $\pi_{\mathsf{bw.god.sm}}$ and $\psi_{\mathsf{bw.god.sm}}$, the compiler of [10] cannot be applied to $\phi_{\mathsf{bw.god.sm}}$ which uses private channels. This holds since if private channels are used, then a party may need to prove different statements to different parties to prove its 'honest behavior' via zero-knowledge. The issue in this approach is that the honest parties at the end of each round will not have consistent view of which parties have aborted / identified to be corrupt. This is crucial as the next round message would depend on it. To bring them to the same page will consume extra rounds which will compromise on the desirable round-preserving property of the compiler of [10]. Thus, we obtain round-optimal protocols by applying the compiler on our broadcast-only protocols i.e $\pi_{\mathsf{bw.god.sm}}$ and $\psi_{\mathsf{bw.god.sm}}$. The former yields a 3-round malicious (god|ua)-BoBW protocol in the CRS model which is a tight upper bound as proven by our lower bound (Theorem 5.6). The latter yields a 2-round (god|ua)-BoBW protocol in the CRS and PKI model which is also round-optimal, as 1-round MPC protocols are known to be impossible for any meaningful security notion ([112]). Notably, the latter demonstrates that our lower bound of Theorem 5.6 can be circumvented in the presence of PKI.

We present the formal description of our 3-round malicious (god|ua)-BoBW protocol $\pi_{\mathsf{bw.god}}$ in $\mathcal{F}_{\mathsf{zk}}$-hybrid model in Figure 5.5 below, where $\mathcal{F}_{\mathsf{zk}}$ denotes the ideal functionality realizing zero-knowledge. In the CRS model, $\mathcal{F}_{\mathsf{zk}}$ can be realized using NIZKs to obtain the 3-round maliciously secure (god|ua)-BoBW MPC protocol. In the private setup model (CRS and PKI), the 2-round malicious (god|ua)-BoBW protocol can be similarly obtained upon compiling the 2-round semi-malicious protocol $\phi_{\mathsf{bw.god.sm}}$.

Protocol $\pi_{\mathsf{bw.god}}$

Inputs: Party P_i has x_i, r_i as input and random input respectively for $i \in [n]$.

Output: $y = f(x_1 \dots x_n)$ or $\perp$

Common Input: The 3-round broadcast-only semi-malicious protocol $\pi_{\mathsf{bw.god.sm}}$ which is parsed as $\{\mathsf{NextMsg}_\ell^k(x_k; r_k; m_1 \dots m_{\ell-1})\}_{\ell\in[3],k\in[n]}$ where $\mathsf{NextMsg}_\ell^k(x_k; r_k; m_1 \dots m_{\ell-1})$ denote the next message function of P_k in Round ℓ, given the messages $m_1, \dots, m_{\ell-1}$ broadcast so far i.e in Rounds 1 to $\ell - 1$. The output computation function of P_k is denoted as $y = \mathsf{Output}_k(x_k, r_k, m_1, m_2, m_3)$. Let $R_{k,\ell}$ be the relation that gets as input $x = (m_1, ..., m_{\ell-1}, m_\ell^k)$ and a witness $w = (x_k, r_k)$, and returns 1 if and only if $\mathsf{NextMsg}_\ell^k(x_k; r_k; m_1 \dots m_{\ell-1}) = m_\ell^k$

Model: $\mathcal{F}_{\mathsf{zk}}$-hybrid model

Protocol steps. For each round ℓ from $\ell = 1$ to 3:

- Let $m_{\ell-1} = m_{\ell-1}^1 \dots m_{\ell-1}^n$ be the concatenation of messages broadcast by the parties in

Round $(\ell - 1)$. (assume $m_0 = \emptyset$).

- Each P_k does the following: Compute $m_\ell^k = \mathsf{NextMsg}_\ell^k(x_k; r_k; m_1 \dots m_{\ell-1})$. Broadcast m_ℓ^k.
- For all $k' \in [n]$, invoke the $\mathcal{F}_{\mathsf{zk}}^{R_{k',\ell}}$ ideal functionality corresponding to the relation $R_{k',\ell}$ on common input $(m_1 \dots m_{\ell-1}, m_\ell^{k'})$. In addition, for $k = k'$, P_k acts as prover and inserts its private input $w = (x_k, r_k)$. If $\mathcal{F}_{\mathsf{zk}}^{R_{k',\ell}}$ returns 0, set $m_\ell^{k'} = \perp$

Output. Let $m_3 = m_3^1 \dots m_3^n$. Each P_k outputs $\mathsf{Output}_k(x_k; r_k; m_1, m_2, m_3)$.

Figure 5.5: 3-round maliciously-secure (god|ua)-BoBW Protocol $\pi_{\mathsf{bw.god}}$

We state the formal theorem below (proof deferred to Section 5.6.4) Assumption wise, our upper bound constructions rely on 2-round semi-malicious oblivious transfer and NIZK in the common random/reference string model upon using the protocols of [93, 35] to realize $\pi_{\mathsf{ua.sm}}$.

Theorem 5.11 *Let (n, s, t) be such that $s + t < n$. Assuming the existence of a 3-round (resp., 2-round with PKI) broadcast-only semi-malicious (god|ua)-BoBW MPC and NIZKs, there exists a 3 (resp., 2)-round MPC protocol in the presence of CRS (resp., CRS and PKI) that simultaneously achieves (i) $\mathcal{F}_{\mathsf{god}}$ (Figure 2.4) when at most $t < n/2$ parties are corrupt and (ii) $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2) when at most $s < n$ parties are corrupt, maliciously in both cases.*

A minor observation is that we can replace the last round broadcast with point-to-point communication in the expense of relaxing ua to sa security in the dishonest majority setting. However, use of broadcast in the earlier rounds is crucial since it enables the honest parties to be in agreement on whether a corrupt party has aborted or not which is crucial to ensure that the output computation is done on a unique set of inputs.

Security with Identifiability. Lastly, since the compiler of [10] uses NIZKs to prove correctness of each round, it offers the property of identifiability. Thus our maliciously-secure (god|ua)-BoBW protocols achieve the stronger notion of identifiable abort in case of dishonest majority, with no extra assumption. Therefore, we obtain the above theorem where $\mathcal{F}_{\mathsf{ua}}$ is replaced with $\mathcal{F}_{\mathsf{idua}}$ (Figure 2.5).

5.5.2 Upper Bound for (god|ua)-BoBW MPC in Plain Model

In this section, we present a 5-round (god|ua)-BoBW protocol in the plain model. For our construction, we resort to the compiler of [35] that transforms any generic $(k-1)$-round delayed-semi-malicious MPC protocol to a k-round malicious MPC protocol for any $k \geq 5$. Our 5-round construction comes in two steps: a) first, we show that our 3-round semi-malicious protocol $\pi_{\mathsf{bw.god.sm}}$ (described in Section 5.5.1.1) is delayed-semi-maliciously secure (refer Section 5.6.5.1

for proof) and then b) we plug in this 3-round BoBW protocol in a modified compiler of [35] that carries over the BoBW guarantees, while the original compiler works for security with abort. Our final 5-round compiled protocol faces several technical difficulties in the proof, brought forth mainly by the need to continue the simulation in case the protocol must result in god, which needs deep and non-trivial redressals. The techniques we use to tackle the challenges in simulation are also useful in constructing a 4-round ($\mathsf{god}|\mathsf{ua}$)-BoBW protocol based on sub-exponentially secure trapdoor permutations and ZAPs. We give a sketch of this construction in Section 5.8 (built upon the protocol of [64]) as a step towards resolving the open question of proving the impossibility or alternately constructing a 4-round ($\mathsf{god}|\mathsf{ua}$)-BoBW protocol under polynomial-time assumptions.

5.5.2.1 The compiler of [35]

Substituting $k = 5$, we recall the relevant details of the compiler of [35] that transforms a 4-round delayed-semi-malicious protocol ϕ_{dsm} to a 5-round maliciously-secure protocol π achieving security with abort. Each party commits to her input and randomness using a 2-round statistically binding commitment scheme Com in the first two rounds. The four rounds of the delayed-semi-malicious protocol ϕ_{dsm} are run as it is in Round $1, 2, 4$ and 5 respectively (Round 3 is skipped) with two additional sets of public-coin delayed-input witness indistinguishable proofs (WI). The first set of proofs (WI^1) which is completed by Round 4, is associated with the first 3 rounds of ϕ_{dsm}. In addition to proving honest behaviour in these rounds, this set of proofs enables the simulator of the malicious protocol to extract the inputs of the corrupt parties, in order to appropriately emulate the adversary for the delayed-semi-malicious simulator in the last but one round. The second set of proofs (WI^2) which is completed by Round 5, is associated with proving honest behaviour in *all* rounds of ϕ_{dsm}. To enable the simulator to pass the WI proofs without the knowledge of the inputs of the honest parties, it is endowed with a *cheat* route (facilitated by the *cheating* statement of the WI proof, while the *honest* statement involves proving honest behaviour wrt inputs committed via Com) which requires the knowledge of the trapdoor of the corrupt parties; which the simulator can obtain by rewinding the last 2 rounds of a trapdoor-generation protocol (Trap) run in the first 3 rounds of the final construction. To enable this cheat route of the simulator, the compiler has an additional component, namely 4-round non-malleable commitment NMCom run in Rounds 1 - 4.

We discuss the tools used in the compiler of [35] in Figure 5.6 and present further details of the compiler in Figure 5.7 below.

Tools used in [35] compiler

- A $(k-1)$-round delayed-semi-malicious protocol ϕ_{dsm} for computing a function f.
- A 2-message statistically binding commitment scheme Com from one-way functions.
- A 3-round protocol Trap to set up a trapdoor between a sender (S) and a receiver (R) as the following sequence of rounds:

  R1: S samples a signing and verification key pair $(\mathsf{sk}, \mathsf{vk})$ of a signature scheme and sends vk to R.

  R2: R sends a random message $m \leftarrow \{0,1\}^\lambda$ to S.

  R3: S computes a signature σ on m using sk and sends σ to R who accepts if (m, σ) is valid w.r.t. vk.

  A valid trapdoor td w.r.t. a verification key vk constitutes of (m, σ, m', σ') such that $m' \neq m$ and σ and σ' are valid signatures of messages m and m' respectively corresponding to vk.
- A 4-round non-malleable commitment scheme NMCom.
- A 4-round public-coin delayed-input witness indistinguishable proof WI.

Figure 5.6: Tools used in [35] compiler

Compiler of [35]

5-round Malicious Protocol π from 4-round delayed-semi-malicious protocol ϕ_{dsm}

Each party $P_i, i \in [n]$ runs the following sub-components with every $P_j, j \in [n] \setminus \{i\}$:

- **Delayed-semi-malicious protocol** ϕ_{dsm}: The 4 messages of ϕ_{dsm} are sent in rounds $(1, 2, 4, 5)$ of π i.e. round 3 of π is skipped in which no messages of ϕ_{dsm} are sent.
- **Commitment** Com: P_i commits to his input and randomness (x_i, r_i) using the commitment protocol Com to P_j. Let the commitment be denoted by $c_{i\to j}$. The two messages of Com are run in the first two rounds of π.
- **Trapdoor generation** Trap: The 3-round trapdoor generation protocol Trap is run in rounds $1-3$ between P_j as the sender and P_i as the receiver. Let $\mathsf{Trap}_{j\to i}$ be the produced transcript and $\mathsf{vk}_{j\to i}$ be the verification key that P_j sends to P_i.
- **Non-Malleable Commitment** NMCom: P_i commits to a random string $s^0_{i\to j}$ to P_j using NMCom in rounds $1-4$. Let $\mathsf{NMCom}_{i\to j}$ denote the produced commitment.

- P_i sends another random string $s^1_{i\to j}$ in the clear to P_j in round 4.

- **First proof of correctness** WI^1: P_i initiates an instance of witness indistinguishable proofs, say $\mathsf{WI}^1_{i\to j}$ in rounds $1-4$ to prove to P_j that he has generated the first 3 messages of ϕ_{dsm} correctly using the input and randomness committed in $c_{i\to j}$. In detail, let $\mathsf{WI}^1_{i\to j}$ denote the proof generated by P_i to P_j to prove correctness of one of the following statements:

  - *Honest Statement:* P_i has correctly generated the first 3 messages of ϕ_{dsm} using the input and randomness committed in $c_{i\to j}$.
  - *Cheating Statement:* XOR of the share $s^0_{i\to j}$ committed to in $\mathsf{NMCom}_{i\to j}$ and the share $s^1_{i\to j}$ is a valid trapdoor w.r.t. verification key $\mathsf{vk}_{j\to i}$.

  Each party P_i verifies all pairwise proofs $\{\mathsf{WI}^1_{i\to j}\}_{i,j\in[N]}$ (proofs are publicly verifiable). If any proof is not accepting, P_i aborts and outputs $\perp$.

- **Second proof of correctness** WI^2: P_i intiates an instance of witness indistinguishable proofs, say $\mathsf{WI}^2_{i\to j}$ in rounds $2-5$ to prove to P_j that he has generated *all* messages of ϕ_{dsm} correctly. In detail, let $\mathsf{WI}^2_{i\to j}$ denote the proof generated by P_i to P_j to prove correctness of one of the following statements:

  - *Honest Statement:* P_i has correctly generated *all* messages of ϕ_{dsm} using the input and randomness committed in $c_{i\to j}$.
  - *Cheating Statement:* XOR of the share $s^0_{i\to j}$ committed to in $\mathsf{NMCom}_{i\to j}$ and the share $s^1_{i\to j}$ is a valid trapdoor w.r.t. verification key $\mathsf{vk}_{j\to i}$.

- **Output Computation:** P_i verifies all proofs i.e.$\{\mathsf{WI}^2_{i\to j}\}_{i,j\in[N]}$. If any proof is not accepting, it aborts and outputs $\perp$. Else, it computes the output according to the underlying delayed-semi-malicious ϕ_{dsm}.

Figure 5.7: Compiler of [35] for $k = 5$

Next, we give an overview of the simulator $\mathcal{S}$ (details appear in [35]) for the 5-round compiled protocol π that uses the simulator $\mathcal{S}_\phi$ of the underlying 4-round protocol ϕ_{dsm}. To emulate the ideal-world adversary corrupting parties in set $\mathcal{C}$, $\mathcal{S}$ invokes the malicious adversary $\mathcal{A}_\pi$ and simulates a real execution of π for $\mathcal{A}_\pi$ by acting on behalf of the honest parties in set $\mathcal{H}$. Recall that the delayed-semi-malicious security of ϕ_{dsm} guarantees that it is secure against an adversary $\mathcal{A}_\phi$ who can choose to behave arbitrarily in the protocol as long as it writes a valid witness (which consists of an input randomness pair $(\{x_i, r_i\}_{i\in\mathcal{C}})$ on behalf of all corrupt parties) on the witness tape of the simulator $\mathcal{S}_\phi$ in the penultimate round such that the witness (x, r) can justify *all* the messages sent by him. In order to avail the services of $\mathcal{S}_\phi$, $\mathcal{S}$ needs to transform

the malicious adversary $\mathcal{A}_\pi$ to a delayed-semi-malicious adversary $\mathcal{A}_\phi$ i.e. it needs a mechanism to write (x, r) on the witness tape of $\mathcal{S}_\phi$. This is enabled via extraction of witness i.e. $\{x_i, r_i\}_{i \in \mathcal{C}}$ from the WI^1 proofs sent by $\mathcal{A}_\pi$ as the prover via rewinding its last two rounds (Round $3, 4$ of π).

Apart from the above set of rewinds for extraction of corrupt parties' inputs, another set of rewinds is required for the following reason: Consider messages of honest parties simulated by $\mathcal{S}_\phi$ that are used by $\mathcal{S}$ to interact with $\mathcal{A}_\pi$ during the execution of π. Here, $\mathcal{S}$ cannot convince $\mathcal{A}_\pi$ in the two sets of WI proofs that these messages are honestly generated. Hence, he opts for the route of the *cheating* statement of the WI proofs which requires the knowledge of the trapdoor of the corrupt parties. At a high-level, Trap (i.e the 3-round trapdoor generation protocol described in Figure 5.6) between a sender S and receiver R allows R to obtain a message-signature pair (m, σ), where σ is computed by S using his signing key sk (corresponding to verification key vk which S sends to R in Round 1) on message m chosen by R (m is sent by R to S in Round 2). The trapdoor of party P_i, consists of *two* valid message-signature pairs with respect to the verification key of P_i. The simulator extracts the trapdoor of parties in $\mathcal{C}$ by rewinding the adversary $\mathcal{A}_\pi$ in Rounds 2 and 3 till he gets an additional valid message-signature pair. The trapdoor has been established this way to ensure that only the simulator (and not the adversary himself) is capable of passing the proofs via the *cheating* statement.

Finally, we point that the two sets of rewinds (Round 2-3 and Round 3-4 of π) can be executed by $\mathcal{S}$ while maintaining that the interaction with $\mathcal{S}_\phi$ is *straight-line* since Round 3 of the compiled protocol is 'dummy' i.e does not involve messages of ϕ_{dsm}. This 'dummy' round is crucial to avoid rewinding of messages in ϕ_{dsm}. Since there are no messages of ϕ_{dsm} being sent in Round 3, $\mathcal{S}$ can simply replay the messages of ϕ_{dsm} (obtained via $\mathcal{S}_\phi$) to simulate Round 2 and Round 4 of π during the rewinds.

5.5.2.2 Our 5-round BoBW construction

Our final goal of a $(\mathsf{god}|\mathsf{ua})$-BoBW protocol $\pi_{\mathsf{bw.god.plain}}$ is obtained by applying the compiler of [35] to our delayed-semi-malicious-secure $(\mathsf{god}|\mathsf{ua})$-BoBW protocol $\pi_{\mathsf{bw.god.sm}}$ (described in Section 5.5.1.1) with slight modifications. Broadly speaking, to preserve the BoBW guarantees from semi-malicious to malicious setting upon applying the compiler, the malicious behaviour of corrupt P_i in the compiled protocol is translated to an analogous scenario when semi-malicious P_i aborts (stops communicating) in the underlying protocol $\pi_{\mathsf{bw.god.sm}}$. Towards this, we make the following modification: Recall from the construction of $\pi_{\mathsf{bw.god.sm}}$ that each party P_i is unanimously assigned a boolean indicator i.e. flag_i by the remaining parties which is initialized to 1 and is later set to 0 if P_i aborts (stops) in the first two rounds. Accounting for malicious

behavior, we now require the value of flag_i to be decided based on not just P_i's decision to abort in a particular round but also on whether he misbehaves in the publicly-verifiable Trap protocol or WI proofs. Specifically, if P_i misbehaves in Trap or the first set of proofs WI^1 with P_i as prover fails, flag_i is set to 0 (analogous to P_i aborting in Round 1 or 2 of $\pi_{\mathsf{bw.god.sm}}$). Further, if the second set of proofs WI^2 with P_i as prover fails, then the last round message of P_i is discarded (analogous to P_i aborting in last round of $\pi_{\mathsf{bw.god.sm}}$).

Next, we point that in our compiled protocol, the 3 rounds of the underlying semi-malicious protocol $\pi_{\mathsf{bw.god.sm}}$ are run in Rounds 1, 4 and 5 respectively. As opposed to compiler of [35] which needed a single 'dummy' round on top of the delayed-semi-malicious protocol, we face an additional simulation technicality (elaborated in the next section) that demands two 'dummy' rounds. This could be enabled while maintaining the round complexity of 5, owing to our 3 (and not 4) round delayed semi-malicious protocol. Furthermore, as described earlier, in order to simulate the WI proofs on behalf of an honest prover towards some corrupt verifier P_i, the simulator requires the knowledge of the trapdoor of P_i which would be possible only if P_i is alive (has not aborted) during the rounds in which trapdoor extraction occurs i.e. Round 2 and Round 3. While the simulator of [35] simply aborts incase any party aborts, the simulator of our BoBW protocol cannot afford to do so as god must be achieved even if upto $t < n/2$ parties abort.

We handle this by adding a supplementary condition in our construction, namely, a prover needs to prove the WI proofs only to verifiers who have been alive until the round in consideration. This completes the description of the modifications of our compiler over [35]. The round-by-round interplay of the different components is given in Table 5.3. We present the detailed description of our 5-round (god|ua)-BoBW MPC protocol $\pi_{\mathsf{bw.god.plain}}$ (incorporating the above modifications) in the plain model in Figure 5.8.

	$\pi_{\mathsf{bw.god.sm}}$	Com	Trap	NMCom	WI^1	WI^2
Round 1	R1	R1	R1	R1	R1	
Round 2		R2	R2	R2	R2	R1
Round 3			R3	R3	R3	R2
Round 4	R2			R4	R4	R3
Round 5	R3					R4

Table 5.3: $\pi_{\mathsf{bw.god.plain}}$

Protocol $\pi_{\mathsf{bw.god.plain}}$

5-round Malicious (god|ua)-BoBW MPC Protocol $\pi_{\mathsf{bw.god.plain}}$ from 3-round delayed-semi-malicious BoBW protocol ϕ_{dsm}

Primitives: Tools mentioned in Figure 5.6 with ϕ_{dsm} instantiated with $\pi_{\mathsf{bw.god.sm}}$ (Figure 5.4).

Round 1. Each party $P_i, i \in [n]$ does the following with $P_j, j \in [n] \setminus \{i\}$:

- Execute Round 1 of ϕ_{dsm}. Initialize $\mathsf{flag}_k = 1$ for all $k \in [n]$ as per ϕ_{dsm}.

- Run Round 1 of $\mathsf{Com}_{i\to j}$ to commit to his input and randomness (x_i, r_i) to P_j. Let the commitment be denoted by $c_{i\to j}$. Run Round 1 of $\mathsf{Com}_{j\to i}$ (where P_j acts as committer) as receiver.
- Run Round 1 of $\mathsf{Trap}_{i\to j}$ as sender, with $\mathsf{vk}_{i\to j}$ denoting the verification key.
- Run Round 1 of $\mathsf{NMCom}_{i\to j}$ as committer and Round 1 of $\mathsf{NMCom}_{j\to i}$ as receiver (with P_j as committer).
- Run Round 1 of $\mathsf{WI}^1_{i\to j}$ as prover and Round 1 of $\mathsf{WI}^1_{j\to i}$ as verifier (with P_j as prover).

Round 2. Each party $P_i, i \in [n]$ does the following with $P_j, j \in [n] \setminus \{i\}$:

- Run Round 2 of $\mathsf{Com}_{i\to j}$ and $\mathsf{Com}_{j\to i}$.
- Run Round 2 of $\mathsf{Trap}_{j\to i}$ (as receiver).
- Run Round 2 of $\mathsf{NMCom}_{i\to j}$ and $\mathsf{NMCom}_{j\to i}$.
- Run Round 2 of $\mathsf{WI}^1_{i\to j}$ and $\mathsf{WI}^1_{j\to i}$. Also, run Round 1 of $\mathsf{WI}^2_{i\to j}$ as prover and Round 1 of $\mathsf{WI}^2_{j\to i}$ as verifier (with P_j as prover).
- Set $\mathsf{flag}_j = 0$ if P_j aborts in Round 1 or Round 2.

Round 3. Each party $P_i, i \in [n]$ does the following with $P_j, j \in [n] \setminus \{i\}$:

- Run Round 3 of $\mathsf{Trap}_{i\to j}$ (as sender).
- Run Round 3 of $\mathsf{NMCom}_{i\to j}$ and $\mathsf{NMCom}_{j\to i}$.
- Run Round 3 of $\mathsf{WI}^1_{i\to j}$ and $\mathsf{WI}^1_{j\to i}$. Also, run Round 2 of $\mathsf{WI}^2_{i\to j}$ and $\mathsf{WI}^2_{j\to i}$.
- Set $\mathsf{flag}_j = 0$ if either P_j aborts in Round 3 or if there exists a $k \in [n], k \neq j$ such that the message-signature pair (m, σ) in $\mathsf{Trap}_{j\to k}$ is not valid w.r.t. $\mathsf{vk}_{j\to k}$. Broadcast enables everyone to agree on this.

Round 4. Each party $P_i, i \in [n]$ does the following with $P_j, j \in [n] \setminus \{i\}$:

- Execute Round 2 of ϕ_{dsm}.
- Run Round 4 of $\mathsf{NMCom}_{i\to j}$ in order to commit to a random string $s^0_{i\to j}$. Run Round 4 of $\mathsf{NMCom}_{j\to i}$ as receiver. Additionally, send another random string $s^1_{i\to j}$ on clear to P_j.
- Run Round 4 of $\mathsf{WI}^1_{j\to i}$ as verifier. If $\mathsf{flag}_j = 1$, run Round 4 of $\mathsf{WI}^1_{i\to j}$ to prove to P_j the correctness of the first 2 messages of ϕ_{dsm}. In detail, $\mathsf{WI}^1_{i\to j}$ proves correctness of one of the following statements: (1) *Honest Statement:* P_i has correctly generated the first 2 messages of ϕ_{dsm} using the input and randomness committed in $c_{i\to j}$. (2) *Cheating*

Statement: XOR of the share $s^0_{i\to j}$ committed to in $\mathsf{NMCom}_{i\to j}$ and the share $s^1_{i\to j}$ is a valid trapdoor $\mathsf{td}_{j\to i}$ w.r.t. verification key $\mathsf{vk}_{j\to i}$.

- Run Round 3 of $\mathsf{WI}^2_{i\to j}$ and $\mathsf{WI}^2_{j\to i}$.
- Set $\mathsf{flag}_j = 0$ if either P_j aborts in Round 4 or if there exists a $k \in [n], k \neq j$ such that $\mathsf{WI}^1_{j\to k}$ leads to *reject*. Public verifiability of WI proofs enables this.

Round 5. Each party $P_i, i \in [n]$ does the following $P_j, j \in [n] \setminus \{i\}$:

- Execute Round 3 of ϕ_{dsm}.
- Run Round 4 of $\mathsf{WI}^2_{j\to i}$ as verifier. If $\mathsf{flag}_j = 1$, run Round 4 of $\mathsf{WI}^2_{i\to j}$ to prove to P_j the correctness of *all* messages of ϕ_{dsm} that he broadcasted. In detail, $\mathsf{WI}^2_{i\to j}$ proves correctness of one of the following statements: (1) *Honest Statement:* P_i has correctly generated *all* messages of ϕ_{dsm} using the input and randomness committed in $c_{i\to j}$ (2) *Cheating Statement:* XOR of the share $s^0_{i\to j}$ committed to in $\mathsf{NMCom}_{i\to j}$ and the share $s^1_{i\to j}$ is a valid trapdoor $\mathsf{td}_{j\to i}$ w.r.t. verification key $\mathsf{vk}_{j\to i}$.
- **Output Computation:** If any proof $\mathsf{WI}^2_{j\to k}$ is not accepting for any $k \in [n], k \neq j$, discard the message from P_j. Compute the output as per ϕ_{dsm}.

Figure 5.8: 5-round Malicious (god|ua)-BoBW MPC Protocol $\pi_{\mathsf{bw.god.plain}}$ from 3-round delayed-semi-malicious BoBW protocol ϕ_{dsm}

5.5.2.3 Proof-sketch for 5-round (god|ua)-BoBW protocol.

The simulator for the compiler of [35] runs in different stages. Plugging it for our 5-round (god|ua)-BoBW construction with appropriate modifications, we present a high-level overview of the simulation. Let $\mathcal{S}_{\mathsf{bw.god.plain}}$ and $\mathcal{S}_{\mathsf{bw.god.sm}}$ denote the simulators corresponding to $\pi_{\mathsf{bw.god.plain}}$ and the underlying delayed semi-malicious protocol $\pi_{\mathsf{bw.god.sm}}$ respectively. Stage 1 involves running the first three rounds with the following changes compared to the real-execution of the protocol: a) Commit to 0 in Com instances (run in Round 1, 2) involving honest party as committer. b) Invoke the simulator for the semi-malicious protocol, $\mathcal{S}_{\mathsf{bw.god.sm}}$ to generate the first message of $\pi_{\mathsf{bw.god.sm}}$ in Round 1 on behalf of honest parties. The rest of the actions in Round 1 - 3 on behalf of honest parties are emulated by $\mathcal{S}_{\mathsf{bw.god.plain}}$ as per protocol specifications. Note that the simulator wrt compiler in [35] proceeds beyond the first stage only when the adversary did not cause an *abort* on behalf of any corrupt party in Stage 1. Else, it aborts. This works out because their protocol promises security with abort and hence, simply terminates if a party aborts. However our protocol, in case of honest majority, promises god with the output being computed on the actual input of the parties who have been alive till last but one round.

To accommodate this, $\mathcal{S}_{\mathsf{bw.god.plain}}$ cannot simply afford to terminate in case a corrupt party aborts. It needs to continue the simulation with respect to corrupt parties who are alive, which demands rewinding. It can thus be inferred that $\mathcal{S}_{\mathsf{bw.god.plain}}$ must always proceed to rewinds unless all the corrupt parties are exposed by adversary in Stage 1. The second and the fourth stage, in particular, are concerned with rewinding of the adversary to enable $\mathcal{S}_{\mathsf{bw.god.plain}}$ to extract some information. In Stage 2, the adversary is reset to the end of Round 1 and Rounds $2, 3$ are rewound in order to enable $\mathcal{S}_{\mathsf{bw.god.plain}}$ to extract trapdoor of corrupt parties. In more detail, consider $\mathsf{Trap}_{j\to i}$ executed between corrupt sender P_j and honest P_i wrt verification key $\mathsf{vk}_{j\to i}$. Now, $\mathcal{S}_{\mathsf{bw.god.plain}}$ acting on behalf of P_i computes the trapdoor of P_j wrt $\mathsf{vk}_{j\to i}$ to be *two* message-signature pairs constituted by one obtained in Stage 1 and the other as a result of rewinding in Stage 2 (note that both signatures are wrt $\mathsf{vk}_{j\to i}$ sent in Round 1 of $\mathsf{Trap}_{j\to i}$; rewinds involve only Round 2, 3). To enable continuation of the simulation after Stage 2, which requires the knowledge of the trapdoors of corrupt parties who are alive, the logical *halt* condition for the rewinds is: *stop when you have enough!* This translates to- stop at the ℓ^{th} rewind if a valid trapdoor has been obtained for the set of corrupt parties alive across the ℓ^{th} rewind. Since the ℓ^{th} (last) rewind is expected to provide one valid (m, σ) pair (i.e message, signature pair) out of two required for the trapdoor, all that is required is for the corrupt party to have been alive across at least one previous rewind. Let the set of parties alive across i^{th} rewind be denoted by $\mathbb{A}_{i+1}$ ($\mathbb{A}_1$ represents the set of parties that were alive in the execution preceeding the rewinds i.e after Stage 1), then the condition formalizes to: halt at rewind ℓ if $\mathbb{A}_{\ell+1} \subseteq \mathbb{A}_1 \cup \cdots \cup \mathbb{A}_\ell$.

While this condition seems appropriate, it leads to the following subtle issue. The malicious adversary can exploit this stopping condition by coming up with a strategy to choose the set of aborting and the alive parties (say, according to some unknown distribution D pre-determined by the adversary) such that the final set of alive parties $\mathbb{A}$ in the transcript output by the simulator (when the rewinds halt) will be biased towards the set of parties that were alive in the earlier rewinds. (Ideally the distribution of the set of alive parties when simulator halts should be identical to D). This would lead to the view output by the simulator being distinguishable from the real view. A very similar subtle issue appears in zero-knowledge (ZK) protocol of [115] - While we defer the details of this issue of [115] to Section 5.6.5.2, we give a glimpse into how their scenario is analogous to ours below. Consider a basic 4-round ZK protocol with the following skeleton: the verifier commits to a challenge in Round 1 which is subsequently decommitted in Round 3. The prover responds to the challenge in Round 4. At a very high-level, the protocol of [115] follows a cut-and-choose paradigm involving N instances of the above basic protocol. Here, the verifier chooses a random subset $S \subset [N]$ of indices and decommits

to the challenges made in those indices in Round 3. Subsequently, the prover completes the ZK protocol for instances with indices in S. The simulator for the zero-knowledge acting on behalf of the honest prover involves rewinds to obtain 'trapdoors' corresponding to the indices in S. However, note that the verifier can choose different S in different rewinds. Therefore, the simulator is in a position to produce an accepting transcript and stop at the ℓ^{th} rewind only when it has trapdoors corresponding to all indices in S chosen by the adversary during the ℓ^{th} rewind. However, if the simulation is stopped at the execution where the above scenario happens for the 'first' time, their protocol suffers an identical drawback as ours. In particular, the malicious verifier can choose the set of indices S in a manner that the distribution of the views output by the simulator is not indistinguishable from the real view. Drawing analogy in a nutshell, the set of indices chosen by the malicious verifier is analogous to the set of alive corrupt parties in our context (details in Section 5.6.5.2). We thereby adopt the solution of [115] and modify our halting condition as: halt at rewind ℓ if $\mathbb{A}_{\ell+1} \subseteq \mathbb{A}_1 \cup \cdots \cup \mathbb{A}_\ell$ **and** $\mathbb{A}_{\ell+1} \nsubseteq \mathbb{A}_1 \cup \cdots \cup \mathbb{A}_{\ell-1}$. [115] gives an elaborate analysis showing why this simulation strategy results in the right distribution. With this change in simulation of Stage 2, the simulation of Stage 3 can proceed identical to [35] which involves simulating the WI^1 proofs via the *fake* statement using the knowledge of trapdoor.

Proceeding to simulation of Stage 4, we recall that the simulator of [35] involves another set of rewinds in Stage 4 which requires to rewind Round 3 and 4 to extract the witness i.e. the inputs and randomness of the corrupt parties from WI^1. Similar to Stage 2, two successful transcripts are sufficient for extraction. Thus, the simulator is in a position to halt at ℓ^{th} rewind if all the corrupt parties that are alive in Stage 4 have been alive across at least one previous rewind. Next, following the same argument as Stage 2, it seems like the *halting* condition for Stage 2 should work, as is, for Stage 4 too. With this conclusion, we stumbled upon another hurdle elaborated in this specific scenario: Recall that the trapdoors extracted for corrupt parties in Stage 2 are used here to simulate the WI^1 proofs (as described in Stage 3). It is thereby required that $\mathcal{S}_{\mathsf{bw.god.plain}}$ already has the trapdoors for the corrupt parties that are alive in Stage 4. Let $\mathbb{T}$ be the set of trapdoors accumulated at the end of Stage 2. Consider a party, say P_i, which stopped participating in Round 3 of the last rewind ℓ of Stage 2 (P_i was alive till Round 2 of ℓ^{th} rewind). $\mathcal{S}_{\mathsf{bw.god.plain}}$ still proceeds to Stage 4 without being bothered about the trapdoor of P_i (as the halting condition is satisfied). However in Stage 4, when the adversary is reset to the end of Round 2 of ℓ^{th} rewind, P_i came back to life again in Round 3. The simulation of WI^1 proofs with P_i as a verifier will be stuck if $\mathbb{T}$ does not contain the trapdoor for P_i. Hence, it is required to accommodate the knowledge of set $\mathbb{T}$ during Stage 4. Accordingly $\mathcal{S}_{\mathsf{bw.god.plain}}$ does the following in Stage 4: During each rewind, if a party (say P_i) whose trapdoor is not

known becomes alive during Round 3, store the signature sent by P_i in Round 3 (as part of Trap) and go back to Stage 2 rewinds (if P_i's trapdoor is still unknown). Looking ahead, storing the signature of P_i ensures that the missing trapdoor of P_i in $\mathbb{T}$ can cause $\mathcal{S}_{\mathsf{bw.god.plain}}$ to revert to Stage 2 rewinds atmost once (if the same scenario happens again i.e P_i becomes alive in Round 3 during Stage 4 rewinds, then another (message, signature) pair wrt verification key of P_i is obtained in this rewind by $\mathcal{S}_{\mathsf{bw.god.plain}}$; totalling upto 2 pairs which suffices to constitute valid trapdoor of P_i which can now be added to $\mathbb{T}$). Else, if $\mathbb{T}$ comprises of the trapdoor of all the corrupt parties that are alive during the rewind of Stage 4, then adhere to the same halting condition as Stage 2. This trick tackles the above described problematic scenario, while ensuring that the simulation terminates in polynomial time and maintains indistinguishability of views.

Before concluding the section, we highlight two important features regarding the simulation of $\pi_{\mathsf{bw.god.plain}}$: Despite the simulator $\mathcal{S}_{\mathsf{bw.god.plain}}$ reverting to Stage 2 rewinds in some cases (unlike the simulation of [35]), the simulation terminates in polynomial-time since this can occur atmost once per corrupt party (as argued above). Lastly, since there is a possibility of reverting back to simulation of Round 2 after simulation of Round 4, we keep an additional 'dummy' Round 2 as well (on top of 'dummy' Round 3 as in [35]) in our construction. This allows us to maintain the invariant that $\mathcal{S}_{\mathsf{bw.god.sm}}$ is never rewound. To be more specific, as there are no messages of underlying semi-malicious protocol being sent in Round 2, 3; even if $\mathcal{S}_{\mathsf{bw.god.plain}}$ needs to return to Stage 2 from Stage 4 (after Round 4 has been simulated by obtaining the relevant message from $\mathcal{S}_{\mathsf{bw.god.sm}}$) and resume the simulation from Stage 2 onwards, the message of $\pi_{\mathsf{bw.god.sm}}$ sent in Round 4 can simply be replayed. We are able to accommodate two dummy rounds while maintaining the round complexity of 5 owing to the privilege that our delayed-semi-malicious protocol is just 3 rounds. This completes the simulation sketch. Assumption wise, our construction relies on 2-round semi-malicious oblivious transfer (a building block of our 3-round delayed-semi-malicious BoBW MPC $\pi_{\mathsf{bw.god.sm}}$). We state the formal theorem below.

Theorem 5.12 *Let (n, s, t) be such that $s + t < n$. Let $\pi_{\mathsf{bw.god.sm}}$ realises– (i) $\mathcal{F}_{\mathsf{god}}$ (Figure 2.4) when at most $t < n/2$ parties are corrupt and (ii) $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2) when at most $s < n$ parties are corrupt, delayed-semi-maliciously in both cases. Then $\pi_{\mathsf{bw.god.plain}}$ in the plain model realises– (i) $\mathcal{F}_{\mathsf{god}}$ when at most $t < n/2$ parties are corrupt and (ii) $\mathcal{F}_{\mathsf{ua}}$ when at most $s < n$ parties are corrupt, maliciously in both cases. It takes 5 rounds, assuming that $\pi_{\mathsf{bw.god.sm}}$ takes 3 rounds.*

Proof: The proof which includes the complete description of the simulator, a discussion about its indistinguishability to the real view and its running time is deferred to Section 5.6.5.3. □

Extension to Identifiability. We additionally point that the publicly-verifiable WI proofs render identifiability to our construction. Thus our maliciously-secure $(\mathsf{god}|\mathsf{ua})$-BoBW protocol achieves the stronger notion of identifiable abort in case of dishonest majority, with no extra assumption. Therefore, we obtain the above theorem where $\mathcal{F}_{\mathsf{ua}}$ is replaced with $\mathcal{F}_{\mathsf{idua}}$ (Figure 2.5).

A minor observation is that we can replace the last round broadcast with point-to-point communication in our $(\mathsf{god}|\mathsf{ua})$-BoBW protocol $\pi_{\mathsf{bw.god.plain}}$ at the expense of relaxing ua to sa security in the dishonest-majority setting.

5.6 Security Proofs

Before presenting the security proofs, we introduce the notion of a semi-malicious adversary, which has been regarded as an effective intermediate step to attain malicious security.

5.6.1 Semi-malicious and Delayed-semi-malicious Security

Semi-malicious security had been introduced in [10] and subsequently used by many works as a stepping-stone for achieving malicious security. We use two variants of semi-malicious security– the original definition of [10, 160] and a variant known as delayed-semi-malicious security [35].

A semi-malicious adversary is modelled as an interactive Turing machine which, in addition to the standard tapes, has a special witness tape. In each round of the protocol, whenever the adversary produces a new protocol message m on behalf of some party P_k, it must also write to its special witness tape some pair (x, r) of input x and randomness r that explains its behavior. More specifically, all of the protocol messages sent by the adversary on behalf of P_k up to that point, including the new message m, must exactly match the honest protocol specification for P_k when executed with input x and randomness r. Note that the witnesses given in different rounds need not be consistent. Also, we assume that the attacker is rushing and hence may choose the message m and the witness (x, r) in each round adaptively, after seeing the protocol messages of the honest parties in that round (and all prior rounds). Lastly, the adversary may also choose to abort the execution on behalf of P_k in any step of the interaction.

Definition 5.3 *We say that a protocol π securely realizes $\mathcal{F}$ for semi-malicious adversaries if it satisfies Definition 2.3 when we only quantify over all semi-malicious adversaries $\mathcal{A}$.*

We point that a party controlled by the semi-malicious adversary must invoke the ideal functionality with either $\perp$ or a valid input in the input phase.

A stronger variant of semi-malicious adversary, denoted as delayed semi-malicious, was introduced in the work of [35]. Informally, a party P_k, under the influence of delayed-semi-malicious

adversary, acts like one under a semi-malicious adversary, except that, it only "explains" all its messages once, before the last round (unlike a semi-malicious party who explains each of its messages after each round). This is formalized by letting P_k write to its special witness tape before the last round some pair (x, r) of input x and randomness r which is required to be consistent with all P_k's messages.

Definition 5.4 *We say that a protocol π securely realizes $\mathcal{F}$ for delayed-semi-malicious adversaries if it satisfies Definition 2.3 when we only quantify over all delayed-semi-malicious adversaries $\mathcal{A}$.*

The real world for delayed-semi-malicious security is defined identically as the real world for semi-malicious security except that adversary $\mathcal{A}$ is only required to provide a witness in the second last round i.e round $L-1$ with respect to a protocol of L rounds. Correspondingly, the ideal world is defined identically as the ideal world for semi-malicious security except that the simulator interacting with the adversary $\mathcal{A}$ (as a black-box) receives the witness that $\mathcal{A}$ output after round $L-1$.

5.6.2 Proof of Security of $\pi_{\mathsf{bw.fair}}$ (Theorem 5.4)

We give a brief intuition of the proof of Theorem 5.4 before presenting the formal proof. First, consider the case of dishonest majority. If $\mathcal{A}$ aborts the computation of $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$, then all honest parties output $\perp$. Suppose $\mathcal{A}$ allows all honest parties to get authenticated t-shares of the output y as output of $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$, then honest parties would either output y or $\perp$ depending on whether $(t+1)$ valid output shares are received in Round $(r+1)$ or not. Unanimity amongst the honest parties follows directly from the argument of Lemma 5.3. Thus we can conclude that $\pi_{\mathsf{bw.fair}}$ achieves ua in case of dishonest majority. Moving on to the honest majority setting, $\mathcal{A}$ again has two choices - whether to allow computation of $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$ to succeed or not. In the former case, since there are $(t+1)$ honest parties, their output shares would suffice to reconstruct the output irrespective of any misbehavior of $\mathcal{A}$ during Round $(r+1)$; leading to output computation by all. In the latter case, since $\mathcal{A}$ has access to only upto t output shares, he learns nothing about the output and all parties output $\perp$. Thus, $\pi_{\mathsf{bw.fair}}$ achieves fn incase of the honest majority setting. This completes the intuition.

We prove the theorem by presenting two separate simulators for the honest and for the dishonest majority case respectively.

Dishonest Majority. Let $\mathcal{A}$ be a malicious adversary controlling s parties in the hybrid-model execution of $\pi_{\mathsf{bw.fair}}$. The simulator $\mathcal{S}_{\mathsf{bw.fair}}^{\mathsf{dm}}$, running an ideal-world evaluation of the

functionality $\mathcal{F}_{\mathsf{ua}}$ (refer Figure 2.2) computing f whose behaviour simulates the behaviour of $\mathcal{A}$ is described in Figure 5.9.

Simulator $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.fair}}$

- Receive $\{x_i\}_{i\in\mathcal{C}}$ sent to $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ in this hybrid execution model. If for any $i \in \mathcal{C}$, x_i is outside of the domain of input, send $\perp$ as output of $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ to $\mathcal{A}$ and send $\perp$ as the input to $\mathcal{F}_{\mathsf{ua}}$ on behalf of the corrupt parties. The simulation is completed in this case. Else invoke $\mathcal{F}_{\mathsf{ua}}$ on behalf of $\mathcal{A}$ with $\{x_i\}_{i\in\mathcal{C}}$ to receive an output value y in return.
- Compute the authenticated t-sharing of value y (Definition 5.2) as done by $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ and send $z_i = \big(a_i(x), \{v_{ij} = (\mathsf{K}_i, a_j(\mathsf{K}_i))\}_{j\in[n]}\big)$ as output of $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ to P_i $(i \in \mathcal{C})$
- If $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.fair}}$ receives `abort` on behalf of $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ from the adversary, it sends the '`abort`' signal to $\mathcal{F}_{\mathsf{ua}}$ on behalf of $\mathcal{A}$. This concludes the simulation for this case.
- If $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.fair}}$ receives `continue` on behalf of $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ from the adversary, it simulates Round $(r+1)$ as follows:
  - Broadcast $a_i(x)$ for each $(i \in \mathcal{H})$ on behalf of honest P_i and receive message $\{a^*_j(x)\}_{j\in\mathcal{C}}$ from the corrupt parties in Round $(r+1)$.
  - Let $\mathcal{C}' \subset \mathcal{C}$ denote the set of indices for which $a^*_j(x) = a_j(x)$. If $|\mathcal{C}'| + |\mathcal{H}| \geq t+1$, then send '`continue`' to $\mathcal{F}_{\mathsf{ua}}$. Else send '`abort`' to $\mathcal{F}_{\mathsf{ua}}$.

Figure 5.9: Simulator $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.fair}}$ for the case of dishonest majority

We argue that the view of $\mathcal{A}$ in the hybrid world and the ideal world is indistinguishable due to the following reason: Observe that the only difference in the ideal world as compared to the hybrid world is in the output computation of the honest parties - In the ideal world, *all* honest parties output y if $|\mathcal{C}'| + |\mathcal{H}| \geq t+1$, where $\mathcal{C}' \subset \mathcal{C}$ is the set of indices such that $a^*_j(x) = a_j(x)$, else they all output $\perp$. In contrast, in the hybrid world, each honest party P_i outputs the output of Rec in which it participates with the output of $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ in Round $(r+1)$. It follows from the argument in Lemma 5.3 that all honest parties would have identical $\mathcal{V}$ sets comprising only of parties in $\mathcal{H}$ and $\mathcal{C}'$, except with probability $\frac{n^2}{|\mathbb{F}|-1}$. Thus, when $|\mathcal{H}| + |\mathcal{C}'| \geq t+1$, for each honest P_i, $|\mathcal{V}_i| \geq t+1$ leading P_i to output y as output of Rec in the hybrid world as well. Similarly, all honest parties would output $\perp$ in both the ideal and the hybrid world when $|\mathcal{H}| + |\mathcal{C}'| < t+1$. Thus the difference between the two worlds occurs with probability atmost $\frac{n^2}{|\mathbb{F}|-1} \approx \epsilon$, which is negligible when $\mathbb{F} = GF(2^\kappa)$, where $\epsilon \geq n^2 2^{-\kappa}$. This completes the proof for the case of dishonest majority.

Honest Majority. Let $\mathcal{A}$ be a malicious adversary controlling t parties in a hybrid-model execution of $\pi_{\mathsf{bw.fair}}$. The simulator $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.fair}}$, running an ideal-world evaluation of the fair functionality $\mathcal{F}_{\mathsf{fair}}$ (refer Figure 2.3) computing f, whose behaviour simulates the behaviour of $\mathcal{A}$, is described Figure 5.10.

Simulator $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.fair}}$

– Receive $\{x_i\}_{i\in\mathcal{C}}$ sent to $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ by $\mathcal{A}$ in this hybrid-execution model. If for any $i \in \mathcal{C}$, x_i is outside of the domain of input, send $\perp$ as output of $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ to $\mathcal{A}$ and send $\perp$ as the input to $\mathcal{F}_{\mathsf{fair}}$ on behalf of the corrupt parties. The simulation is completed in this case. Else it does the following. Noticeably, in this case $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.fair}}$ cannot call $\mathcal{F}_{\mathsf{fair}}$ yet with the inputs, as the adversary can still abort the protocol by signaling $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ with `abort` in which case all parties will obtain $\perp$ as the output.

  - Choose t random shares $\{s_i\}_{i\in\mathcal{C}} \in \mathbb{F}$. For each $i \in \mathcal{C}$, compute MAC-polynomial $a_i(x)$ that authenticates s_i (Definition 5.1) with corresponding verification information as $v_{ji} = (\mathsf{K}_j, a_i(\mathsf{K}_j))$ of verifier P_j.
  - For each $i \in \mathcal{C}, j \notin \mathcal{C}$, set $v_{ij} = (\mathsf{K}_i, \mathsf{T}_{ij})$ where $\mathsf{K}_i, \mathsf{T}_{ij}$ are sampled randomly from $\mathbb{F}$.
  - Send $\big(a_i(x), \{v_{ij}\}_{j\in[n]}\big)$ as output of $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ to P_i $(i \in \mathcal{C})$.

– If $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.fair}}$ receives `abort` on behalf of $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ from the adversary, it sends $\perp$ as the adversary's input to $\mathcal{F}_{\mathsf{fair}}$. This completes the simulation for this case.

– If $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.fair}}$ receives `continue` on behalf of $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ from the adversary, it sends the inputs $\{x_i\}_{i\in\mathcal{C}}$ to $\mathcal{F}_{\mathsf{fair}}$ and receives output y in return. Recall that $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.fair}}$ had sent $\big(a_i(x), \{v_{ij}\}_{j\in[n]}\big)$ as output of $\mathcal{F}^{\mathsf{sh}}_{\mathsf{ua}}$ to P_i $(i \in \mathcal{C})$ where $v_{ij} = (\mathsf{K}_i, \mathsf{T}_{ij})$ for $j \notin \mathcal{C}$. $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.fair}}$ does the following to simulate Round $(r+1)$:

  - Interpolate a degree-t polynomial $p(x)$ satisfying $p(0) = y$ and $p(i) = a_i(0)$ for $i \in \mathcal{C}$. Set $s'_j = p(j)$ for $j \notin \mathcal{C}$.
  - Corresponding to each $j \notin \mathcal{C}$, interpolate a degree-t polynomial $a_j(x)$ satisfying $a_j(0) = s'_j$ and $a_j(\mathsf{K}_i) = \mathsf{T}_{ij}$ for $i \in \mathcal{C}$.
  - Broadcast $a_j(x)$ on behalf of each honest P_j $(j \notin \mathcal{C})$ in Round $(r+1)$. Output y on behalf of all honest parties.

Figure 5.10: Simulator $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.fair}}$ for the case of honest majority

We now claim that the view of $\mathcal{A}$ in the hybrid world and the ideal world is indistinguishable due to the following: The difference between the hybrid and the ideal execution is that when

$\mathcal{A}$ receives $(a_i(x), \{v_{ij}\}_{j\in[n]})$ for each $i \in \mathcal{C}$ as output from $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$, the values v_{ij} in the former are computed as verification information of the authenticated t-shares of the output y (Definition 5.2) (i.e $v_{ij} = (\mathsf{K}_i, a_j(\mathsf{K}_i))$ with $a_j(0) = p(j)$ as a t-share of y), while in the latter they are random for $j \notin \mathcal{C}$. It is easy to verify that the indistinguishability follows since $\mathcal{A}$ has access to atmost t points on the degree t polynomial $a_j(x)$ for $j \notin \mathcal{C}$. Finally, in the case when $\mathcal{A}$ allows honest parties to obtain the output shares from $\mathcal{F}_{\mathsf{ua}}^{\mathsf{sh}}$, it is easy to check that since $|\mathcal{V}_i| \geq t+1$ for each honest P_i (as there are atleast $(t+1)$ honest parties), each P_i would proceed to reconstruction. Furthermore, the argument made in Lemma 5.3 shows that all honest parties would exclude js from their $\mathcal{V}$ sets such that P_j broadcast the incorrect MAC polynomial corresponding to its output share, except with negligible probability. Subsequently, the correct secret y would be reconstructed. We can thus conclude that all honest parties obtain output y in both the ideal and the hybrid execution, except with negligible probability. This completes the proof for the honest majority setting. This completes the proof of Theorem 5.4.

5.6.3 Proof of Security of $\pi_{\mathsf{bw.god.sm}}$ (Theorem 5.8)

We prove the theorem by demonstrating that the 3-round protocol $\pi_{\mathsf{bw.god.sm}}$ (Figure 5.4) obtained by compiling a 2-round semi-malicious protocol $\pi_{\mathsf{ua.sm}}$ satisfies the security guarantees of $(\mathsf{god}|\mathsf{ua})$-BoBW. We give the description of two simulators, namely $\mathcal{S}_{\mathsf{bw.god.sm}}^{\mathsf{dm}}$ and $\mathcal{S}_{\mathsf{bw.god.sm}}^{\mathsf{hm}}$ that simulates the view of the real-world adversary $\mathcal{A}$ in case of s semi-malicious corruptions and t semi-malicious corruptions respectively. Both $\mathcal{S}_{\mathsf{bw.god.sm}}^{\mathsf{dm}}, \mathcal{S}_{\mathsf{bw.god.sm}}^{\mathsf{hm}}$ internally use the simulator of the semi-malicious protocol $\pi_{\mathsf{ua.sm}}$, say $\mathcal{S}_{\mathsf{ua.sm}}$. The simulator of the adaptive garbling scheme $\mathcal{S}_{\mathsf{ad}}$ is also invoked (Refer Section 2.4.1.1 for details).

The simulator $\mathcal{S}_{\mathsf{bw.god.sm}}^{\mathsf{dm}}$ is described in Figure 5.11. We argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{ua}}, \mathcal{S}_{\mathsf{bw.god.sm}}^{\mathsf{dm}}} \overset{c}{\approx} \text{REAL}_{\pi_{\mathsf{bw.god.sm}}, \mathcal{A}}$ when the semi-malicious adversary $\mathcal{A}$ corrupts $s < n$ parties. The views are shown to be indistinguishable via a series of intermediate hybrids.

Simulator $\mathcal{S}_{\mathsf{bw.god.sm}}^{\mathsf{dm}}$

Round 1. $\mathcal{S}_{\mathsf{bw.god.sm}}^{\mathsf{dm}}$ does the following-

- *Interaction with $\mathcal{S}_{\mathsf{ua.sm}}$ to receive Round 1 of $\pi_{\mathsf{ua.sm}}$:* Execute the simulator $\mathcal{S}_{\mathsf{ua.sm}}(1^\kappa)$ to obtain $\{\mathcal{B}_i^1\}_{i\in\mathcal{H}}$.
- On behalf of each $i \in \mathcal{H}$, setup $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\kappa)$ and broadcast $(\mathsf{pk}_i, \mathcal{B}_i^1)$.
- Receive $\{\mathsf{pk}_j, \mathcal{B}_j^1\}$ broadcast by P_j where $j \in \mathcal{C}$ along with its "witness" (x_j^1, r_j^1) from its witness tape.

Round 2. $\mathcal{S}_{\mathsf{bw.god.sm}}^{\mathsf{dm}}$ does the following-

- For each $i \in \mathcal{H}$: run $(\widetilde{\mathsf{C}}_i, \widetilde{d}_i) \leftarrow \mathcal{S}_{\mathsf{ad}}(1^\lambda, \theta(C_i), 0)$, where θ is the side information known about C_i i.e the topology of the circuit and broadcast $(\widetilde{\mathsf{C}}_i, \widetilde{d}_i)$ on behalf of P_i.
- On behalf of each P_i ($i \in \mathcal{H}$): For each $b \in \{0,1\}$, $k \in [n]$, $j \in \mathcal{C}$, sample $\mathsf{lab}_{i,j}^{k,b}$ at random. For $j \in \mathcal{C}$ broadcast $c_{i,j}^{k,b} = \mathsf{Enc}(\mathsf{pk}_j, \mathsf{lab}_{i,j}^{k,b})$. For $j \notin \mathcal{C}$, broadcast $c_{i,j}^{k,b}$ as encryption of a dummy message.
- For each $j \in \mathcal{C}$: Receive C_j and $\{c_{j,i}^{k,b}\}_{k\in[n],b\in\{0,1\},i\in[n]}$ along with its "witness" (x_j^2, r_j^2) from its witness tape. For $i \in \mathcal{H}$, compute $\mathsf{lab}_{j,i}^{k,b} = \mathsf{Dec}(\mathsf{sk}_i, c_{j,i}^{k,b})$

Round 3. $\mathcal{S}_{\mathsf{bw.god.sm}}^{\mathsf{dm}}$ does the following-

- *Interaction with $\mathcal{S}_{\mathsf{ua.sm}}$ to send Round 1 of $\pi_{\mathsf{ua.sm}}$:* For $j \in \mathcal{C}$, if P_j did not abort in Round 1 or Round 2 of $\pi_{\mathsf{bw.god.sm}}$, use the "witness" (x_j^2, r_j^2) of the corrupt P_j from its witness tape and forward the witness and $\mathcal{B}_j^1$ to $\mathcal{S}_{\mathsf{ua.sm}}$ as the Round 1 message from P_j. Set $(x_j^*, r_j^*) = (x_j^2, r_j^2)$ and $\mathsf{flag}_j = 1$. Else, forward the default values (x_j', r_j') and $\mathcal{B}_j'^1$ computed using the default values to $\mathcal{S}_{\mathsf{ua.sm}}$ as the Round 1 message from P_j. In this case, set $(x_j^*, r_j^*) = (x_j', r_j')$ and $\mathsf{flag}_j = 0$.
- *Invoking the ideal functionality $\mathcal{F}_{\mathsf{ua}}$:* Invoke $\mathcal{F}_{\mathsf{ua}}$ computing f with the set of values $\{x_j^*\}_{j\in\mathcal{C}}$ on behalf of $\mathcal{A}$ and obtain the output y. This is provided to $\mathcal{S}_{\mathsf{ua.sm}}$ as the response from its ideal functionality when invoked by $\mathcal{S}_{\mathsf{ua.sm}}$.
- *Interaction with $\mathcal{S}_{\mathsf{ua.sm}}$ to receive Round 2 of $\pi_{\mathsf{ua.sm}}$*: Invoke $\mathcal{S}_{\mathsf{ua.sm}}$ to obtain $\{\mathcal{B}_i^2\}_{i\in\mathcal{H}}$.
- Set $\mathsf{flag}_i = 1$ for all $i \in \mathcal{H}$. For each $i \in \mathcal{H}$: Run $(\{\mathsf{lab}_i^{k,\mathsf{flag}_k}\}_{k\in[n]}) \leftarrow \mathcal{S}_{\mathsf{ad}}(\mathcal{B}_i^2, 1)$. For each $k \in [n]$, interpolate a degree-s polynomial $M_i^k(x)$ satisfying $M_i^k(0) = \mathsf{lab}_i^{k,\mathsf{flag}_k}$ and $M_i^k(j) = \mathsf{lab}_{i,j}^{k,\mathsf{flag}_k}$ for $j \in \mathcal{C}$ (chosen in Round 2), where $|\mathcal{C}| \leq s$. For $j \in \mathcal{H}$, set $\mathsf{lab}_{i,j}^{k,\mathsf{flag}_k} = M_i^k(j)$.
- For each $i \in [n]$: For $j \in \mathcal{H}$, broadcast $\mathsf{lab}_{i,j}^{k,\mathsf{flag}_k}$. For $j \in \mathcal{C}$, receive $\mathsf{lab}_{i,j}^{k,\mathsf{flag}_k}$.

- *Interaction with $\mathcal{S}_{\mathsf{ua.sm}}$ to send Round 2 of $\pi_{\mathsf{ua.sm}}$*: For $j \in \mathcal{C}$ such that $\mathsf{flag}_j = 1$, use the shares broadcast in Round 3 to reconstruct the labels associated with C_j. If the reconstruction of all labels is successful, proceed to evaluation of C_j and obtain b_j^2 as per the protocol. Send witness (x_j^*, r_j^*) and b_j^2 as Round 2 message to $\mathcal{S}_{\mathsf{ua.sm}}$ from P_j. Else, abort P_j. For $j \in \mathcal{C}$ such that $\mathsf{flag}_j = 0$, compute default b_j^2 as per the protocol and send the default witness (x_j', r_j') and b_j^2 as Round 2 message to $\mathcal{S}_{\mathsf{ua.sm}}$ from P_j.

Output to honest parties: Let $\mathcal{C}' \subset \mathcal{C}$ denote the set of parties controlled by $\mathcal{A}$ who do not abort throughout $\pi_{\mathsf{bw.god.sm}}$. If $|\mathcal{C}'| + |\mathcal{H}| \geq s+1$, $\mathcal{S}_{\mathsf{bw.god.sm}}$ invokes $\mathcal{F}_{\mathsf{ua}}$ computing f with `continue` on behalf of $\mathcal{A}$. Output y on behalf of the honest parties. Else $\mathcal{S}_{\mathsf{bw.god.sm}}$ invokes $\mathcal{F}_{\mathsf{ua}}$ with `abort` on behalf of $\mathcal{A}$ and output $\perp$ on behalf of the honest parties.

Figure 5.11: Description of Simulator $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$

- HYB_0: Same as $\text{REAL}_{\pi_{\mathsf{bw.god.sm}},\mathcal{A}}$.

- HYB_1: Same as HYB_0, except that for $i,j \in \mathcal{H}$, the ciphertext $c^{k,b}_{i,j}$ (for all $k \in [n], b \in \{0,1\}$) broadcast in Round 2 is an encryption of dummy message.

- HYB_2: Same as HYB_1, except that for $i \in \mathcal{H}$, $(\mathsf{C}_i, \{\mathsf{lab}_i^{k,\mathsf{flag}_k}\}_{k\in[n]}, d_i)$ is computed as $(\mathsf{C}_i, d_i) \leftarrow \mathcal{S}_{\mathsf{ad}}(1^\lambda, \theta(C_i), 0)$ and $(\{\mathsf{lab}_i^{k,\mathsf{flag}_k}\}_{k\in[n]}) \leftarrow \mathcal{S}_{\mathsf{ad}}(\mathcal{B}_i^2, 1)$.

- HYB_3: Same as HYB_2 except that $\{\mathcal{B}_i^1, \mathcal{B}_i^2\}_{i\in\mathcal{H}}$ is generated via the simulator $\mathcal{S}_{\mathsf{ua.sm}}$ of the underlying semi-malicious protocol $\pi_{\mathsf{ua.sm}}$.

- HYB_4: Same as HYB_3 except that honest parties output $\perp$ if $|\mathcal{C}'| + |\mathcal{H}| < s+1$, where $\mathcal{C}' \subset \mathcal{C}$ is the set of parties controlled by $\mathcal{A}$ that do not abort throughout $\pi_{\mathsf{bw.god.sm}}$.

Since $\text{HYB}_4 := \text{IDEAL}_{\mathcal{F}_{\mathsf{ua}},\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}}$, we show that every two consecutive hybrids are computationally indistinguishable which completes the proof for the case of s corruptions.

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The difference between the hybrids is that the ciphertext $c^{k,b}_{i,j}$ (for $k \in [n], b \in \{0,1\}$) broadcast in Round 2 using key pk_j for $i,j \in \mathcal{H}$, is the encryption of P_j's share of the encoded input $\mathsf{lab}_i^{k,b}$ i.e $\mathsf{lab}_{i,j}^{k,b}$ in HYB_0 while it is the encryption of a dummy message in HYB_1. The messages in Round 3 by P_i ($i \in \mathcal{H}$) remain the same. The indistinguishability follows via reduction to the security of the public-key encryption scheme ($\mathcal{A}$ has no information about sk_j).

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The difference in the hybrids is the way $(\mathsf{C}_i, \{\mathsf{lab}_i^{k,\mathsf{flag}_k}\}_{k\in[n]}, d_i)$ is computed for $i \in \mathcal{H}$. In HYB_1, it is computed as $(\mathsf{C}_i, e_i, d_i) \leftarrow \mathsf{Gb}(1^\lambda, C_i)$ and then as $\{\mathsf{lab}_i^{k,\mathsf{flag}_k} \leftarrow \mathsf{En}(e_i, \mathsf{flag}_k)\}_{k\in[n]}$. On the other hand, in HYB_2, it is computed as $(\mathsf{C}_i, d_i) \leftarrow \mathcal{S}_{\mathsf{ad}}(1^\lambda, \theta(C_i), 0)$ and $(\{\mathsf{lab}_i^{k,\mathsf{flag}_k}\}_{k\in[n]}) \leftarrow \mathcal{S}_{\mathsf{ad}}(\mathcal{B}_i^2, 1)$. The indistinguishability follows via reduction to the adaptive privacy of the garbling scheme.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The difference between the hybrids is that the values $\{\mathcal{B}_i^1, \mathcal{B}_i^2\}$ for $i \in \mathcal{H}$ are generated using honest parties' inputs in HYB_2 but generated via the simulator $\mathcal{S}_{\mathsf{ua.sm}}$ in HYB_3. The indistinguishability follows directly from the semi-malicious security of the protocol $\pi_{\mathsf{ua.sm}}$.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The difference between the hybrids is that while the honest parties output $\perp$ in HYB_3 if any reconstruction fails, they do so in HYB_4 if $|\mathcal{C}'| + |\mathcal{H}| < s+1$, where $\mathcal{C}' \subset \mathcal{C}$ is

the set of parties controlled by $\mathcal{A}$ that do not abort throughout $\pi_{\mathsf{bw.god.sm}}$. It is easy to check that the difference occurs only when some party in $\mathcal{C}$, say P_j, does not abort in Round 3, but sends an incorrect share, say s' leading to problems in the reconstruction. However, note that the semi-malicious P_j needs to be consistent with the transcript of Round 2 comprising of ciphertexts encrypting the correct share, say s, with his public key pk_j. Thus, the share s' sent by P_j in Round 3 must be a valid decryption of the ciphertext broadcast in Round 2. It now follows from the correctness of the public-key encryption scheme that both s, s' cannot be valid decryptions of the same ciphertext.

This completes the proof of security for the case of $s < n$ corruptions.

The simulator $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ for the case of $t < n/2$ corruptions is described in Figure 5.12. The steps are almost same as that of $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$, and only differs in terms of output computation of the honest parties. We argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}} \overset{c}{\approx} \text{REAL}_{\pi_{\mathsf{bw.god.sm}},\mathcal{A}}$ when the semi-malicious adversary $\mathcal{A}$ corrupts $t < n/2$ parties. The views are shown to be indistinguishable via a series of intermediate hybrids.

Simulator $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$

Round 1. Same as Round 1 of $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ (Figure 5.11).

Round 2. Same as Round 2 of $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ (Figure 5.11).

Round 3. Similar to Round 3 of $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ (Figure 5.11) except the following:

- The ideal functionality $\mathcal{F}_{\mathsf{god}}$ is invoked on behalf of $\mathcal{A}$ instead of $\mathcal{F}_{\mathsf{ua}}$.
- *Additional step:* For each P_j ($j \in \mathcal{C}$), such that $\mathsf{flag}_j = 1$: For each $k \in [n]$, use $\{\mathsf{lab}_{j,i}^{k,\mathsf{flag}_k}\}_{i \in \mathcal{H}}$ to reconstruct $\mathsf{lab}_j^{k,\mathsf{flag}_k}$ (recall that $|\mathcal{H}| = s + 1$). Evaluate C_j using $\{\mathsf{lab}_j^{k,\mathsf{flag}_k}\}_{k \in [n]}$ to obtain b_j^2.

- *Interaction with $\mathcal{S}_{\mathsf{ua.sm}}$ to send Round 2 message of $\pi_{\mathsf{ua.sm}}$*: For $j \in \mathcal{C}$ such that $\mathsf{flag}_j = 1$, send witness (x_j^*, r_j^*) and b_j^2 as Round 2 message to $\mathcal{S}_{\mathsf{ua.sm}}$ from P_j. For $j \in \mathcal{C}$ such that $\mathsf{flag}_j = 0$, compute default b_j^2 as per the protocol and send the default witness (x_j', r_j') and b_j^2 as Round 2 message to $\mathcal{S}_{\mathsf{ua.sm}}$ from P_j.

Output of honest parties: Output y on behalf of all honest parties.

Figure 5.12: Description of Simulator $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$

- HYB_0: Same as $\text{REAL}_{\pi_{\mathsf{bw.god.sm}},\mathcal{A}}$.

- $\text{HYB}_1, \text{HYB}_2, \text{HYB}_3$: Same as $\text{HYB}_1, \text{HYB}_2, \text{HYB}_3$ described previously corresponding to $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$

- HYB_4: Same as HYB_3 except that honest parties do not output $\perp$.

Since $\text{HYB}_4 := \text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}}$, we show that every two consecutive hybrids are computationally indistinguishable. The argument for $\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$ suffices to complete the proof for the case of t corruptions as the indistinguishability of $\text{HYB}_0 \overset{c}{\approx} \text{HYB}_3$ has been described previously in the context of $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The difference between the hybrids is that in HYB_3, honest parties output $\perp$ if any reconstruction fails, but in HYB_4, honest parties do not output $\perp$. The indistinguishability follows as in HYB_3, the $(n-t) = (s+1)$ honest parties would broadcast their correct shares in Round 3 which would suffice for the purpose of successful reconstruction of the s-shared value. Also, as argued earlier, the shares broadcast by non-aborting semi-malicious parties in Round 3 must also be correct. This holds since the semi-malicious parties must remain consistent with the Round 2 message that includes ciphertexts encrypting the correct shares (follows from the correctness of the public-key encryption scheme).

This completes the proof of Theorem 5.8.

5.6.4 Proof of Security of $\pi_{\mathsf{bw.god}}$ (Theorem 5.11)

We prove the theorem by claiming that the protocol $\pi_{\mathsf{bw.god}}$ achieves god against $t < n/2$ malicious corruptions and security with ua against $s < n$ malicious corruptions in the $\mathcal{F}_{\mathsf{zk}}$-hybrid model. For contradiction, assume a malicious adversary $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god}}$ controlling a subset of $t < n/2$ parties, say $\mathcal{C}$, that breaches security of $\pi_{\mathsf{bw.god}}$. We build a semi-malicious adversary $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ corrupting the same set of parties $\mathcal{C}$ for the 3-round semi-malicious BoBW MPC protocol $\pi_{\mathsf{bw.god.sm}}$ as follows. $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ internally uses $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god}}$ and interacts with the honest parties in an execution of $\pi_{\mathsf{bw.god.sm}}$ as follows:

- In each round ℓ ($\ell \in [3]$), $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ forwards the messages received in the execution of $\pi_{\mathsf{bw.god.sm}}$ from the honest parties to $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god}}$. Receive m^i_ℓ from each $P_i (i \in \mathcal{C})$ sent by $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god}}$ in the execution of $\pi_{\mathsf{bw.god}}$.

- Simulate the $\mathcal{F}_{\mathsf{zk}}$ functionality for each Round ℓ ($\ell \in [3]$) as follows: When an honest party should be the prover, just check that the adversary sends the correct statement and return 1 as the response of $\mathcal{F}_{\mathsf{zk}}$. In case where a corrupted party $P_i (i \in \mathcal{C})$ is the prover, check that indeed $\mathsf{NextMsg}^i_\ell(x_i; r_i; m_1 \dots m_{\ell-1}) = m^i_\ell$, where (x_i, r_i) is P_i's witness received by $\mathcal{F}_{\mathsf{zk}}$. Incase this holds, return 1 to $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god}}$, update the witness tape of $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ to include

(x_i, r_i) and send m^i_ℓ on behalf of P_i to honest parties in the execution of $\pi_{\mathsf{bw.god.sm}}$. Incase of failure, abort the party P_i.

- $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ outputs whatever $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god}}$ outputs.

Similarly, using the simulator $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ for $\pi_{\mathsf{bw.god.sm}}$ (refer Theorem 5.8), we can build a simulator $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god}}$ for $\pi_{\mathsf{bw.god}}$ for the honest majority case. Since $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ behaves the same way as $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god}}$, any attack by $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god}}$ controlling $t < n/2$ parties that breaks the security of $\pi_{\mathsf{bw.god}}$ is translated to an attack by $\mathcal{A}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ controlling $t < n/2$ parties to break security of $\pi_{\mathsf{bw.god.sm}}$. This leads to a contradiction as $\pi_{\mathsf{bw.god.sm}}$ achieves god incase of $t < n/2$ semi-malicious corruptions as proved in Theorem 5.8. Similarly, a malicious adversary $\mathcal{A}^{\mathsf{dm}}_{\mathsf{bw.god}}$ for $\pi_{\mathsf{bw.god}}$ controlling a subset of $s < n$ parties, can be used to build a semi-malicious adversary $\mathcal{A}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ corrupting $s < n$ parties that breaks security of $\pi_{\mathsf{bw.god.sm}}$ which is a contradiction. This completes the proof of our claim that $\pi_{\mathsf{bw.god}}$ gives the necessary BoBW security guarantees stated in Theorem 5.11 in the $\mathcal{F}_{\mathsf{zk}}$-hybrid model. In the CRS model, $\mathcal{F}_{\mathsf{zk}}$ can be realized using NIZKs; thereby completing the proof of Theorem 5.11.

5.6.5 Proof of Security of $\pi_{\mathsf{bw.god.plain}}$

Before presenting the proof, we first show that the 3-round protocol $\pi_{\mathsf{bw.god.sm}}$ (Figure 5.4) satisfies the stronger notion of delayed-semi-malicious security (Section 5.6.1) and recall the relevant technicalities in [115] which are useful for our proof.

5.6.5.1 Proof of Delayed-semi-malicious Security

Recall that the delayed-semi-malicious adversary is similar to semi-malicious adversary, except that it is required to provide a witness only in the second-last round. We argue that $\pi_{\mathsf{bw.god.sm}}$ achieves the desired BoBW security guarantees even against such an adversary due to the following: First, we note that the simulators $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ and $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ (Figure 5.11,Figure 5.12) do not require the adversary's witness at the end of Round 1 to simulate Round 2 and use only the witness (x^2_j, r^2_j) output by a corrupt P_j at the end of Round 2 for simulation. Thus, the simulation can proceed identical to $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ and $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ in case of a delayed-semi-malicious adversary who provides witness only during Round 2 (second-last round). Next, we observe that arbitrary malicious behavior in Round 1 by a delayed-semi-malicious adversary does not affect simulation of Round 2 as it involves communication of only adaptive garbled circuits and ciphertexts corresponding to shares of labels of the garbled circuit (encrypted with the appropriate public-key of the share's recipient). It is easy to check from description of the simulators (in Figure 5.11-Figure 5.12) that the simulation of adaptive garbled-circuits requires

only the circuit topology which is independent of the adversary's potentially malicious Round 1 message. Lastly, a malformed public-key sent by an adversary in Round 1 does not affect the simulation as the shares of honest parties are encrypted with their respective well-formed public keys. This misbehavior would only affect the ciphertexts comprising of adversary's share which are simulated identical to the real-world. We point that since the ciphertext is decrypted only in Round 3 after the delayed-semi-malicious adversary provides a witness justifying the well-formedness of its public key, there is no scope of breach in security even if adversary misbehaves in Round 1. We can thus conclude that the simulators $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ and $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ maintain that the adversary's view in the ideal and real-world is indistinguishable even in the face of a delayed-semi-malicious adversary.

5.6.5.2 Recalling [115]

We begin with a quick overview of the 4-round Zero-knowledge argument of [115] that compiles 3-round sigma protocols of the following special form: The prover simply relies on commitments to generate its first round message and decommits to some subset of the commitments depending on the challenge provided by the verifier. Additionally, special soundness guarantee is needed (for details refer to [115]). To amplify soundness of this 3-round zero-knowledge argument system, the entire protocol can be repeated in parallel, where the verifier commits to all the parallel challenges in a first round of the protocol while decommitting in the third round. To avoid malleability attacks by corrupt prover (who can use the verifier's commitment in first round to change it to another commitment that can be open to a valid accepting response depending on the decommitment provided by the verifier in the third round), an approach used is to ask the prover to prove "knowledge" of the messages in its commitment before the verifier decommits its challenge. This can be achieved via extractable commitment schemes which is a commitment scheme with 'proof of knowledge' property. To design a 4-round ZK argument system, [115] follow a cut-and-choose paradigm. Their protocol comprises of N parallel instances of the basic 4-round protocol. In Round 3, the verifier chooses a random $S \subset [N]$ of some size T and decommits to the challenges made in those indices while providing a challenge for the extractable commitment for repetitions outside S. Then in Round 4, the prover will complete the zero-knowledge protocol for the parallel executions with indexes in S and respond to the proof-of-knowledge challenge for the extractable commitment for the remaining indexes. This completes the skeleton of the protocol.

We now elaborate on the simulation technicality relevant to us. To prove zero-knowledge, a simple strategy for the simulator is to obtain the challenge, i.e. "trapdoor" for the indexes in S, rewind and setup the prover messages in such a way that will allow for it to cheat in all instances

corresponding to indices in S. Now, the simulator can conclude with an accepting transcript if the verifier opens the same set S. However, the verifier can choose to reveal different subsets in different "rewindings". However, in any rewinding, either the simulator has succeeded in cheating in all the indexes of the subset revealed by the verifier or has learned a new trapdoor. The natural simulation strategy is as above i.e the simulator tries to extract trapdoors and outputs the "first" accepting transcript when it has managed to cheat in all indexes in the revealed subset. This simple idea however has a subtle flaw. The issue is that one can come up with a strategy for a malicious verifier where the distribution of the views output by the simulator is not indistinguishable from the real view. Roughly speaking, the distribution of the subset S in the transcript output by the simulator will be biased towards indexes revealed earlier in the rewindings. The main technical contribution of [115] is to determine the "stopping" condition for the simulator that will result in the right distribution. Let S_i denote the subset output by adversary in iteration i. The work of [115] proves that the following simulation strategy achieves the goal of maintaining indistinguishability between the view output by the simulator and the real-world view. In any iteration j, if $S_j \subseteq S_1 \cup S_2 \ldots S_{j-1}$, then halt if $S_j \nsubseteq S_1 \cup S_{j-2}$; else proceed to the next iteration.

Next, we give a brief insight into the proof of indistinguishability between the real and simulated view as in [115]. Let $\mathcal{S}_{\mathsf{zk}}$ define the simulator following the simulation strategy outlined above. The following intermediate hybrids are defined:

- H1: In this experiment, the view of the verifier when it interacts with the honest prover with witness ω is considered.
- H2: In this experiment, a simulator $\mathcal{S}^1_{\mathsf{zk}}$ is defined that proceeds with the rewinding strategy as simulator $\mathcal{S}_{\mathsf{zk}}$ does, with the exception that the prover's messages are generated according to the honest prover's strategy. The view output by $\mathcal{S}^1_{\mathsf{zk}}$ is considered here.
- H3: The ideal-world view output by simulator $\mathcal{S}_{\mathsf{zk}}$.

Indistinguishability among each pair of hybrids is proven in [115] to complete the indistinguishability argument.

In the context of simulation of our 5-Round (god|ua)-BoBW MPC construction $\pi_{\mathsf{bw.god.plain}}$, we face a similar scenario as [115] during Stage 2 and Stage 4 rewinds. The set of indices S is analogous to the set of corrupt parties that are alive. We therefore incorporate the halting condition of [115] in our simulation strategy.

5.6.5.3 Security Proof (Theorem 5.12)

Next, we discuss our simulator for the dishonest-majority setting, $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.plain}}$ in Figure 5.13. Note that $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ (Figure 5.11) is the underlying semi-malicious simulator which is invoked in the dishonest majority case. The simulator for honest majority $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god.plain}}$ is same as $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.plain}}$ except that s is replaced by t (in the number of iterations in Stage 2,4 of simulation) and the underlying semi-malicious simulator invoked is $\mathcal{S}^{\mathsf{hm}}_{\mathsf{bw.god.sm}}$ (Figure 5.12). The major differences in our simulator as compared to the simulator of [35] are in Stage 2 and Stage 4 to tackle the challenges that arise due to the required BoBW guarantees.

Simulator $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.plain}}$

Let $\mathcal{C} \subset [n]$ and $\mathcal{H} = [n] \setminus \mathcal{C}$ denote the set of indices of s corrupt parties and the indices of honest parties respectively. The simulation proceeds in stages as follows:

Stage 1: This stage simulates Rounds 1, 2 and 3 of the main thread as follows:

- Invoke $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ to simulate honest parties' messages corresponding to Round 1 of $\pi_{\mathsf{bw.god.sm}}$ (sent in Round 1 of $\pi_{\mathsf{bw.god.plain}}$). Note that Round 1-3 of $\pi_{\mathsf{bw.god.plain}}$ involves only first round of $\pi_{\mathsf{bw.god.sm}}$, to simulate which $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ doesn't need any witness.
- Commitment $c_{i\to j}$ is simulated as follows: If P_i is honest, commit to 0 in $c_{i\to j}$, and if P_j is honest, emulate the receiver of Com honestly.
- Act as the honest receiver of Trap: Upon receiving verification key $\mathsf{vk}_{j\to i}$, send a random challenge message on behalf each honest P_i and receive the corrresponding signature from P_j. Act as honest sender wrt $\mathsf{Trap}_{i\to j}$
- Commit in the first three messages of $\mathsf{NMCom}_{i\to j}$ to a random share $s^0_{i\to j}$.
- Act according to the protocol in the first three messages of $\mathsf{WI}^1_{i\to j}$ (on behalf of honest P_i as prover), $\mathsf{WI}^1_{j\to i}$ (on behalf of honest P_i as verifier) and similarly, first two messages of $\mathsf{WI}^2_{i\to j}$, $\mathsf{WI}^2_{j\to i}$.

Stage 2: This stage involves rewinding Rounds 2 and 3 to extract trapdoors. Let $\mathbb{T}_c, c \in \mathcal{C}$ be a set that contains at most two tuples where each tuple is a set of message-signature pairs for each honest party i.e. $(m_{h\to c}, \sigma_{c\to h})_{h\in\mathcal{H}}$ valid with respect to $\mathsf{vk}_{c\to h}$. Initialize $\mathbb{T}_c = \emptyset$. Let $\mathbb{T}$ be the set of corrupt parties $\{P_i\}$ for which the trapdoor has been obtained i.e. $|\mathbb{T}_i| = 2$.
Let the set of corrupt parties alive after Stage 1 of $\mathcal{S}_{\mathsf{bw.god.plain}}$ be $\mathbb{A}_1$ and $\mathbb{A}_0 = \emptyset$. For each $P_c \in \mathbb{A}_1$, add one tuple to $\mathbb{T}_c$ as follows: $\mathbb{T}_c = \mathbb{T}_c \cup \{(m_{h\to c}, \sigma_{c\to h})_{h\in\mathcal{H}}\}$ where $m_{h\to c}$ is Round 2 message sent by simulator and $\sigma_{c\to h}$ is Round 3 message received by simulator on behalf of each honest party

$P_h, h \in \mathcal{H}$ during Stage 1.

Let the set of corrupt parties alive across i^{th} rewind (iteration) be $\mathbb{A}_{i+1}$. For iterations $\ell = 1$ to $s+1$, the simulator proceeds as follows:

- For both Rounds 2 and 3, on behalf of each honest party in $\mathcal{H}$, simulate all the components $\mathsf{Com}, \mathsf{Trap}, \mathsf{NMCom}, \mathsf{WI}^1, \mathsf{WI}^2$ exactly as in Stage 1.

- Let the set of corrupt parties alive upto Round 3 in this iteration be $\mathbb{A}'$. For each alive party P_c i.e. $P_c \in \mathbb{A}'$, if $|\mathbb{T}_c| < 2$, update $\mathbb{T}_c$ as follows: $\mathbb{T}_c = \mathbb{T}_c \cup \{(m_{h\to c}, \sigma_{c\to h})_{h\in\mathcal{H}}\}$ where $m_{h\to c}$ is Round 2 message sent by simulator and $\sigma_{c\to h}$ is Round 3 message received by simulator on behalf of each honest party $P_h, h \in \mathcal{H}$. If $|\mathbb{T}_c| = 2, \mathbb{T} = \mathbb{T} \cup \{P_c\}$.

Consider the exhaustive cases:

Case *a.* $\mathbb{A}' \nsubseteq \mathbb{A}_1 \cup \cdots \cup \mathbb{A}_\ell$: This implies that a party became alive for the first time in this iteration and the simulator does not have his trapdoor required to proceed to the next stage. The simulator sets $\mathbb{A}_{\ell+1} = \mathbb{A}'$ and continues to the next iteration. Note that every iteration results in adding a tuple to $\mathbb{T}_c$ for some c such that $P_c \notin \mathbb{T}$. Hence, at the end of s iterations $|\mathbb{T}_c| \geq 1$ for each $c \in \mathcal{C}$ must hold. Therefore, the number of iterations is bounded by $(s+1)$ since in that iteration, the simulator will definitely be able to obtain trapdoor wrt all corrupt parties that are alive (by combining the tuple in $\mathbb{T}_c$ with the tuple it obtains in the last iteration before halting).

Case *b.* $\mathbb{A}' \subseteq \mathbb{A}_1 \cup \cdots \cup \mathbb{A}_\ell$ **and** $\mathbb{A}' \subseteq \mathbb{A}_1 \cup \cdots \cup \mathbb{A}_{\ell-1}$: Ignore this case and rewind again i.e. go to Step 1. Note that the simulator has enough trapdoors to proceed to the next stage but this case is still ignored to handle the situation where the adversary can choose the set of alive parties such that the views in the real and the simulated world become distinguishable.

Case *c.* $\mathbb{A}' \subseteq \mathbb{A}_1 \cup \cdots \cup \mathbb{A}_\ell$ **and** $\mathbb{A}' \nsubseteq \mathbb{A}_1 \cup \cdots \cup \mathbb{A}_{\ell-1}$: This is the *halting* condition, when the set of alive parties seen is covered by the set of alive parties seen in the previous ℓ iterations but is not covered by the set of alive parties seen in the first $\ell - 1$ iterations. The simulator sets $\mathbb{A}_{\ell+1} = \mathbb{A}'$ and proceeds to the next stage.

Stage 3: This stage involves simulation of Round 4 of the main thread using trapdoors as follows:

- Invoke $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ to simulate honest parties' messages corresponding to Round 2 of $\pi_{\mathsf{bw.god.sm}}$ (sent in Round 4 of $\pi_{\mathsf{bw.god.plain}}$)

- Simulate Round 3 messages of $\mathsf{WI}^2_{i\to j}$ and $\mathsf{WI}^2_{j\to i}$ (where P_i is prover and verifier respectively) honestly as per the protocol.

- Simulate Round 4 of $\mathsf{WI}^1_{j\to i}$ (on behalf of P_i as verifier) honestly.

- In instances $\mathsf{WI}^1_{i\to j}$ where P_i is an honest prover, do the following: (a) Commit in the last message of $\mathsf{NMCom}_{i\to j}$ to the random share $s^0_{i\to j}$ tossed in Stage 1. (b) Send the other share $s^1_{i\to j} = s^0_{i\to j} \oplus \mathsf{td}_{j\to i}$ on clear, where $\mathsf{td}_{j\to i}$ comprises of the two message-signature pairs wrt $\mathsf{vk}_{j\to i}$ obtained from $\mathbb{T}_j$ wrt honest P_i (c) Prove in the last message of $\mathsf{WI}^1_{i\to j}$ the fake statement that $\mathsf{NMCom}_{i\to j}$ commits to $s^0_{i\to j}$ such that, $\mathsf{td}_{j\to i} = s^1_{i\to j} \oplus s^0_{i\to j}$ is a valid trapdoor w.r.t. $\mathsf{vk}_{j\to i}$.

Stage 4: This stage involves rewinding Rounds 3 and 4 to extract input of corrupt parties from WI^1. Let sets $\mathbb{T}_c, c \in \mathcal{C}$ and $\mathbb{T}$ be defined as in Stage 2. Let the set of corrupt parties alive after Stage 3 of $\mathcal{S}_{\mathsf{bw.god.plain}}$ be $\mathbb{C}_1$ and $\mathbb{C}_0 = \emptyset$. Let the set of corrupt parties alive upto Round 4 of the i^{th} rewind (iteration) be $\mathbb{C}_{i+1}$. For iteration $\ell = 1$ to $s+1$, the simulator proceeds as follows:

1. For Round 3, simulate components $\mathsf{Trap}, \mathsf{NMCom}, \mathsf{WI}^1, \mathsf{WI}^2$ on behalf of each honest party in $\mathcal{H}$ as done in main thread.

2. Let the set of corrupt parties alive in Round 3 be denoted by $\mathbb{B}$. Consider the cases:

   Case *a*. $\mathbb{B} \subseteq \mathbb{T}$: This corresponds to the case when the trapdoors collected so far are sufficient to continue with this iteration. The simulator proceeds to step 3.

   Case *b*. $\mathbb{B} \nsubseteq \mathbb{T}$: This corresponds to the case when there exists at least one additional party (say P_c) that became alive in this iteration for which the simulator does not have the trapdoor. For each such P_c, update $\mathbb{T}_c$ as follows: $\mathbb{T}_c = \mathbb{T}_c \cup \{(m_{h\to c}, \sigma_{c\to h})_{h\in\mathcal{H}}\}$ where $m_{h\to c}$ is round 2 message sent by simulator and $\sigma_{c\to h}$ is round 3 message received by simulator on behalf of each honest party $P_h, h \in \mathcal{H}$. If $|\mathbb{T}_c| = 2, \mathbb{T} = \mathbb{T} \cup \{P_c\}$. Consider two sub-cases:

   Sub-case *b*1. $\mathbb{B} \subseteq \mathbb{T}$: This corresponds to the case when for each P_c, $\mathbb{T}_c$ already contained one message-signature pair and the other message-signature pair collected in this iteration yields trapdoor of P_c i.e. $\mathbb{T}$ now includes P_c. Proceed to step 3.

   Sub-case *b*2. $\mathbb{B} \nsubseteq \mathbb{T}$: This corresponds to the case when this was the first time P_c was alive in Round 3 i.e. $\mathbb{T}_c$ was initially empty. Hence, the one message-signature pair obtained in this iteration is not enough to compute the trapdoor and proceed. Re-run Stage 2 and Stage 3.

3. For Round 4, replay honest parties' message of $\pi_{\mathsf{bw.god.sm}}$ (obtained via $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ in Stage 3) and simulate the third message of WI^2 as in the main thread. Note that we arrive at this step after making sure that we possess the trapdoors for all the alive parties. Simulate the fourth round of NMCom and WI^1 using the trapdoors in $\mathbb{T}$.

4. Let the set of corrupt parties alive be $\mathbb{C}'$. Consider the exhaustive cases:

Case *a.* $\mathbb{C}' \nsubseteq \mathbb{C}_1 \cup \cdots \cup \mathbb{C}_\ell$: This implies that a party became alive for the first time and the simulator can't extract that party's witness in this iteration. The simulator sets $\mathbb{C}_{\ell+1} = \mathbb{C}'$ and continues to the next iteration.

Case *b.* $\mathbb{C}' \subseteq \mathbb{C}_1 \cup \cdots \cup \mathbb{C}_\ell$ **and** $\mathbb{C}' \subseteq \mathbb{C}_1 \cup \cdots \cup \mathbb{C}_{\ell-1}$: Ignore this case and rewind again i.e. go to step 1. Note that the simulator had enough executions to extract the witness to proceed to the next stage but this case is still ignored to handle the issue where the adversary can choose the set of alive parties in a manner that views in the real and the simulated world become distinguishable.

Case *c.* $\mathbb{C}' \subseteq \mathbb{C}_1 \cup \cdots \cup \mathbb{C}_\ell$ **and** $\mathbb{C}' \nsubseteq \mathbb{C}_1 \cup \cdots \cup \mathbb{C}_{\ell-1}$: This is the *halting* condition, when the set of alive parties seen is covered by the set of alive parties seen in the previous ℓ iterations but is not covered by the set of alive parties seen in the first $\ell - 1$ iterations. The simulator sets $\mathbb{C}_{\ell+1} = \mathbb{C}'$. For each corrupt $P_j \in \mathbb{C}_{\ell+1}$, let $k < \ell$ be the iteration in which P_j was alive i.e. $P_j \in \mathbb{C}_{k+1}$. Use iterations k and ℓ to extract the input, randomness (x_j, r_j) as done in [35] i.e from the two accepting transcripts in iterations k, ℓ that share the same first two messages of $\mathsf{WI}^1_{j\to i}$ with P_j as prover and P_i as honest verifier. Proceed to next stage.

Stage 5: Using the corrupted parties' inputs and random tapes $\{x_j, r_j\}$ extracted (corresponding to $P_j \in \mathbb{C}'$ i.e corrupt parties who have been alive upto Round 4 in the final iteration of Stage 4), simulate honest parties' messages in Round 5 as follows:

- Feed $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ the witness $\{x_j, r_j\}$ for $P_j \in \mathbb{C}'$ and default values (x'_j, r'_j) for $P_j \in \mathcal{C} \setminus \mathbb{C}'$. Use $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ to simulate the honest parties' message in the last round.
- In instances $\mathsf{WI}^2_{i\to j}$ where P_i is an honest prover, prove in the last message of $\mathsf{WI}^2_{i\to j}$ the fake statement that $\mathsf{NMCom}_{i\to j}$ commits to $s^0_{i\to j}$ such that, $\mathsf{td}_{j\to i} = s^1_{i\to j} \oplus s^0_{i\to j}$ is a valid trapdoor w.r.t. $\mathsf{vk}_{j\to i}$. Simulate $\mathsf{WI}^2_{j\to i}$ with P_i as verifier honestly.
- For each $P_j \in \mathbb{C}'$ such that all proofs $\mathsf{WI}^2_{j\to k}$ are accepting for $k \in [n]$, send Round 3 message of $\pi_{\mathsf{bw.god.sm}}$ on behalf of P_j to $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$. If $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ invokes its ideal functionality with `abort` (resp., `continue`), $\mathcal{S}_{\mathsf{bw.god.plain}}$ invokes its ideal functionality $\mathcal{F}_{\mathsf{ua}}$ with `abort` (resp., `continue`).

Figure 5.13: Description of simulator $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.plain}}$

We argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{ua}},\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.plain}}} \overset{c}{\approx} \text{REAL}_{\pi_{\mathsf{bw.god.plain}},\mathcal{A}}$ when the malicious adversary $\mathcal{A}$ corrupts $s < n$ parties. We also need to prove that the simulator runs in expected polynomial time. Consider the following series of intermediate hybrids, most of which are similar to the series of hybrids in [35]. While most of the security arguments follow from [35] and [115], the crux of our proof lies in Claim 5.8. This claim argues that inspite of our modification in Stage 4 simulation where we re-run Stage 2 onwards in some cases, the simulator continues to run in

expected polynomial time as the number of re-runs occur only a fixed constant number of times in the worst case.

- HYB_0: Same as $\text{REAL}_{\pi_{\mathsf{bw.god.plain}},\mathcal{A}}$.

- HYB_1: Same as HYB_0, except that after generating the first 3 messages, Round 2 and 3 are rewound for extraction of trapdoors according to the Stage 2 simulation strategy in Figure 5.13 (with the difference that the components $\mathsf{Com}, \mathsf{NMCom}, \mathsf{WI}^1, \mathsf{WI}^2$ are done on honest inputs).

- HYB_2: Same as HYB_1 except that in Round 4 of the main thread, for every honest party P_h and every alive corrupt party P_c, share $s^1_{h\to c}$ is set to $s^0_{h\to c} \oplus \mathsf{td}_{c\to h}$ where $\mathsf{td}_{c\to h}$ is the trapdoor w.r.t. $\mathsf{vk}_{c\to h}$.

- HYB_3: Same as HYB_2 except that in WI^1 and WI^2 of the main thread, for every honest party P_h as a prover and every alive corrupt party P_c as verifier, P_h proves the *cheating* statement that $\mathsf{NMCom}_{h\to c}$ commits to $s^0_{h\to c}$ such that $s^0_{h\to c} \oplus s^1_{h\to c} = \mathsf{td}_{c\to h}$ which is a valid trapdoor w.r.t. $\mathsf{vk}_{c\to h}$.

- HYB_4: Same as HYB_3 except that after generating Round 4 message, Round 3 and 4 are rewound for extraction of witness from WI^1 according to the Stage 4 simulation strategy in Figure 5.13 (with the difference that the Com and messages of the underlying delayed semi-malicious protocol are done on honest inputs and randomness).

- HYB_5: Same as HYB_4 except that every honest party P_h commits to 0 in $c_{h\to i}$ $(i \neq h)$.

- HYB_6: Same as HYB_5 except that the messages of underlying delayed-semi-malicious protocol $\pi_{\mathsf{bw.god.sm}}$ are simulated using $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$.

Note that $\text{HYB}_6 := \text{IDEAL}_{\mathcal{F}_{\mathsf{ua}},\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.plain}}}$. To complete the proof for s corruptions, we prove two things for each hybrid: a) it runs in expected polynomial time b) it is indistinguishable from the previous hybrid in the sequence. Proving a) for the last hybrid implies that the simulator also runs in expected polynomial time.

Claim 5.1 $\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$

Proof: First, we note that the components of the compiler are run in an identical manner in both HYB_0 and HYB_1. To argue indistinguishability, we need to prove the following: the distribution on the set of corrupt parties that are alive in the view output by the simulation

strategy of Stage 2 when run with honest inputs, is identical to the same distribution in the real-world execution of the protocol. The argument follows similar to *Claim 3.2* of [115] (which proves indistinguishability of $\mathsf{H1}$ and $\mathsf{H2}$ as defined in Section 5.6.5.2). □

Claim 5.2 HYB_1 *runs in expected polynomial time.*

Proof: To argue that HYB_1 runs in polynomial time, we need to prove that the simulation strategy of Stage 2 (run with honest inputs) is such that the expected running time of the iterations / rewinds (that are executed till the halting condition is satisfied) is polynomial. The proof follows from the argument of *Claim 3.4* of [115] (which argues that the expected running time of $\mathcal{S}^1_{\mathsf{zk}}$ as defined in Section 5.6.5.2 is polynomial).

□

Claim 5.3 $\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$

Proof: The argument follows directly from the proof of *Claim 10.11* in [35] (via reduction to hiding of NMCom). □

Claim 5.4 HYB_2 *runs in expected polynomial time.*

Proof: Same as proof of *Claim 10.10* in [35]. □

Claim 5.5 $\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$

Proof: The argument follows directly from the proof of *Claim 10.14* in [35] (via reduction to witness indistinguishability property of the WI proofs). □

Claim 5.6 HYB_3 *runs in expected polynomial time.*

Proof: Same as proof of *Claim 10.13* in [35]. □

Claim 5.7 $\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$

Proof: The difference between HYB_3 and HYB_4 is that HYB_4 has an additional set of rewinds according to the simulator's strategy in Stage 4 (except that it is run with honest inputs). The proof of this claim is similar to argument in Claim 5.1. The only difference is that the rewinds in Stage 4 may involve reverting to Stage 2 rewinds in certain cases. However, this does not interfere with the indistinguishability argument as it suffices to argue that the final view output by the simulation strategy in Stage 4 (after possibly reverting and restarting from Stage 2 until a point when Stage 4 is simulated without any callbacks to Stage 2) is indistinguishable to the view in HYB_3. □

Claim 5.8 HYB_4 *runs in expected polynomial time.*

Proof: Firstly, we note that Stage 4 rewinds in HYB_4 have additional possible calls to Stage 2 rewinds. Barring those calls, the Stage 4 rewinds are similar to Stage 2; hence they take expected polynomial time as argued in Claim 5.2. Also, individually each additional Stage 2 call takes expected polynomial time as discussed in the run-time of HYB_1. We can thus conclude that if the number of possible Stage 2 calls is bounded by a constant (predefined parameter of the protocol), then Claim 5.8 is automatically implied. We analyze the number of calls to Stage 2 below.

Recall that Stage 2 rewinds can be called internally from an iteration of Stage 4 in the following condition con: a party (say P_i) whose trapdoor is not known i.e. $P_i \notin \mathbb{T}$ becomes alive in Round 3 of that iteration. The simulator first adds the pair (m_i, σ_i) obtained w.r.t. P_i to $\mathbb{T}_i$. He still could be at most one pair away from obtaining his trapdoor which is the case when the Stage 2 rewinds are actually called. Observe that the Stage 2 rewinds are never called again w.r.t P_i because the mere occurrence of condition con is sufficient to serve another (m_i, σ_i) pair to the simulator and 2 such pairs are enough to compose the trapdoor of P_i. Hence, the upper bound on the number of additional Stage 2 calls per corrupt party is 1. Since there are at most s corrupt parties, this bounds the number of additional calls to s; hence completing the proof. □

Claim 5.9 $\text{HYB}_4 \overset{c}{\approx} \text{HYB}_5$

Proof: The difference between HYB_4 and HYB_5 is that while Com with honest party as committer is run with respect to honest party's input (and randomness) in the former, the latter involves commitment to 0 in the main thread and all the rewinds. The claim can be proven similar to *Claim 3.6* of [115] (that argues indistinguishability between $\mathsf{H2}$ and $\mathsf{H3}$ as defined in Section 5.6.5.2) - Let there exist a polynomial $p(n)$ such that for infinitely many $n's$, HYB_4 and HYB_5 can be distinguished with probability $\frac{1}{p(n)}$. Consider the truncated experiments $\overline{\text{HYB}_4}$ and $\overline{\text{HYB}_5}$ which proceed exactly as HYB_4 and HYB_5 respectively with the exception that the simulation is aborted if it runs more than $np(n)t(n)$ steps where $t(n)$ is the polynomial that bounds the expected run-time of HYB_4. By an averaging argument (similar to [115]), it is possible to distinguish $\overline{\text{HYB}_4}$ and $\overline{\text{HYB}_5}$ with probability at least $\frac{1}{2p(n)}$.

Similar to [115], we consider a series of intermediate hybrids $\overline{\text{HYB}_4^0}, \ldots, \overline{\text{HYB}_4^s}$ where in each hybrid $\overline{\text{HYB}_4^\ell}$, the strategy of HYB_5 (i.e. commit to 0 in Com) is followed in first ℓ iterations of the Stage 2 rewinds and the strategy of HYB_4 (i.e. commit to honest input and randomness in Com) is followed in the remaining iterations. Also if $\overline{\text{HYB}_4^\ell}$ runs over $np(n)t(n)$ steps, the

simulator outputs $\perp$. Note that $\overline{\text{HYB}_4^0} = \overline{\text{HYB}_4}$ and $\overline{\text{HYB}_4^s} = \overline{\text{HYB}_5}$. If $\overline{\text{HYB}_4}$ and $\overline{\text{HYB}_5}$ are distinguishable by probability $\frac{1}{2p(n)}$, then there exists an index i such that $\overline{\text{HYB}_4^i}$ and $\overline{\text{HYB}_4^{i+1}}$ are distinguishable by probability $\frac{1}{2np(n)}$ (taking upper bound on s to be n). Now, the distinguisher used to distinguish between $\overline{\text{HYB}_4^i}$ and $\overline{\text{HYB}_4^{i+1}}$ can be used to break the hiding property of Com (argument similar to *Claim 10.20* in [35]). □

Claim 5.10 HYB_5 *runs in expected polynomial time.*

Proof: The only difference between HYB_4 and HYB_5 is in the value committed in Com, which does not change the run-time. Hence the proof follows from the claim discussing the run-time of HYB_4. □

Claim 5.11 $\text{HYB}_5 \overset{c}{\approx} \text{HYB}_6$

Proof: The argument for the claim follows similar to the argument in Claim 5.9. We consider a similar series of sub-hybrids and argue that indistinguishability of HYB_5 and HYB_6 boils down to the indistinguishability between a consecutive pair of sub-hybrids. Now, the indistinguishability of a consecutive pair of sub-hybrids follows from the security of the delayed semi-malicious simulator $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ (similar to *Claim 10.23* in [35]). □

Claim 5.12 HYB_6 *runs in expected polynomial time.*

Proof: The only difference between HYB_5 and HYB_6 is in the way the messages of $\pi_{\mathsf{bw.god.sm}}$ are generated. Hence the proof follows from the claim discussing the run-time of HYB_5 and the knowledge that $\mathcal{S}^{\mathsf{dm}}_{\mathsf{bw.god.sm}}$ runs in expected polynomial time. □

5.7 Appendix: MPC with ua security

In this section, we discuss in detail how to augment the security of the existing 4-round MPC protocols of [113, 15, 60] from sa to ua.

5.7.1 Boosting security of [113] to ua

This section is organised as follows: After a brief informal overview of the protocol of [113], we first highlight the manner in which the adversary could disrupt unanimity and our proposed fixes to tackle the issues. Next, for the sake of completeness, we recall the original protocol of [113]. Lastly, we present the modified protocol that incorporates the fixes and achieves ua.

5.7.1.1 Issues in boosting security of [113] to ua

We begin with a high-level sketch of the protocol of [113]. The Boolean circuit, corresponding to the function f to be computed, is first made resilient to additive attacks by applying the AMD transformations of [97, 98] and then the BMR randomized encoding [19] is applied on the transformed function. As per BMR encoding, each party P_i ($i \in [n]$) picks two keys $k^i_{w,0}, k^i_{w,1}$ and a bit λ^i_w for every wire w, the latter as its contribution to a mask bit λ_w for w. The garbled table of each 2-input gate g with inputs wires a, b and output c comprises of 4 rows (for the 4 input combinations). The $(\alpha, \beta)^{\text{th}}$ row of a NAND gate consists of n ciphertexts, where the i^{th} ciphertext encrypts the $b^{\text{th}}_{\alpha,\beta}$ output key from P_i's contribution on wire c, namely $k^i_{c,b_{\alpha,\beta}}$ where $b_{\alpha,\beta} = \mathsf{NAND}(\lambda_a \oplus \alpha, \lambda_b \oplus \beta) \oplus \lambda_c = [(\lambda_a \oplus \alpha)\dot{(}\lambda_b \oplus \beta) \oplus 1] \oplus \lambda_c$. This clever encoding enables evaluating the circuit in masked form where the actual bits blinded with corresponding mask (λ) bits alone get published. Starting with input bits blinded with their masks, these garbled tables enable to compute blinded output bits. Specifically, the keys corresponding to the masked bits for the input wires a, b of a gate are used to decrypt the relevant n keys for the output wire, namely k^i_{c,δ_c}, where δ_c denotes the masked bit on the output wire c. Each P_i deduces the value of δ_c by comparing the key obtained from decryption of ith ciphertext with its pair of keys $(k^i_{c,0}, k^i_{c,1})$. For the output gates, the mask value λ is given out as output translation table to recover the actual output.

Notably, the BMR encoding i.e. every ciphertext in the garbled tables represents a degree-3 monomial over parties' random inputs. To compute the monomials, [113] gives a 3-round protocol π_{poly} (building upon the 3-bit multiplication protocol of [3]) against "defensible" adversary (i.e adversary volunteers a defense or explanation of its actions so far, consisting of some inputs and randomness at the end of Round 3). The protocol ends with every party having an XOR-share of the encoding (every ciphertext of the garbled tables), XOR-share of the output translation tables and the masked input bits. Now, to compile this defensible protocol to a malicious one, 2-round witness indistinguishable proofs (derived from ZAPs) are used whose "witness" would act as the "defense". Once all the actions upto Round 3 are verified via ZAPs, all parties broadcast their respective shares in Round 4 to reconstruct the garbled tables, which can now be locally evaluated and decoded (using BMR decoding first and AMD decoding subsequently) to obtain the output. Note that there is no proof of correctness for Round 4, meaning that the adversary $\mathcal{A}$ can modify the output translation tables, arbitrarily making the honest parties output the wrong answer. This is tackled by taking additionally as input a MAC key, say K_i, from each party P_i, and augmenting the output of the MPC to include the authentication of the function output y under each of the parties' keys. In more

detail, $\{y, t_1 \dots t_n\}$ denotes the output received by each party at the end of the protocol where $t_i = \mathsf{MAC}(\mathsf{K}_i, y)$ for $i \in [n]$. Now, an honest party P_i would accept the output only if t_i validates y as per its private MAC key K_i. The privacy of K_i for an honest P_i, makes it hard for the adversary to change y and match it with a valid t_i.

Another technicality arises, as the ZAPs in [113] fall short of guarding against an adversary that can lead to encryption of bit-strings in the garbled table that are not entirely the relevant output wire keys, but rather mix of bits from both keys. This would help the adversary learn some bits of the other key after decoding. This leakage is controlled via a slight variant of BMR encoding where the garbled tables encrypt random values unrelated to the actual keys for the wires and the keys are given out in a blinded format using blinders derived from the random values operated with pairwise independent hash functions. Now, even if the adversary learns some bit of the other random value, the left-over hash lemma ensures that the other blinder is still random, guarding the privacy of the other key. This completes the high-level description of the protocol.

There are two ways the adversary can disrupt unanimity of [113], that stem from the specifics of BMR encoding and decoding. To present these issues comprehensively, we abstract out the BMR encoding and decoding in Figure 5.14 and the backbone protocol of [113] in Figure 5.15, stripping the ZAPs and other related details. We describe the issues and elaborate the solutions below.

Issue I: Selective manipulation of the output and MAC. Though the MAC mechanism on the output y keeps the sanity of y, the dedicated and independent MAC for every party P_i makes it easy for an adversary to selectively tweak some MACs and create disagreement. A corrupt P_j, by broadcasting a modified share of the output translation table λ_w^j for an output wire w during Round 4, can make sure that AMD-encoding of $(y, t_1, \dots, t_i', \dots, t_n)$ is reconstructed, where t_i' is the only tampered MAC. Now an honest P_i output $\perp$, while the rest output y leading to a disagreeing honest population.

Unanimity in this case is enforced by making C output y that is authenticated using the authentication with public verifiability introduced in Section 5.3.1. Specifically, the additional private input to C on behalf of P_i is now the verification information v_i, which is a pair of uniformly-picked from $\mathbb{F}$ secret points (K_i, y_i) (see Definition 5.1). The output of C is $a(x)$ where $a(x)$ is the n-degree MAC polynomial with $a(0) = y$ and $a(\mathsf{K}_i) = y_i$ for $i \in [n]$. An $\mathcal{A}$ trying to change the output to the AMD-encoding of $a^*(x) \neq a(x)$ would be detected by *each* honest P_i except with negligible probability since v_i is unknown to him.

Lemma 5.8 *When y is authenticated using the above form of authentication, all the parties*

either output y or $\perp$, except with probability at most $\frac{n}{|\mathbb{F}|-1}$.

Proof: Assuming $\mathcal{H}$ is the set of honest parties, the adversary can make the honest parties disagree by guessing one of the keys K of the honest parties so that it helps reconstruct $a^*(x) \neq a(x)$ that verifies to only the honest party holding the guessed K value. The probability of the above event is $\frac{|\mathcal{H}|}{|\mathbb{F}|-1} < \frac{n}{|\mathbb{F}|-1}$. With $\mathbb{F} = GF(2^\kappa)$, the above probability is negligible in κ. □

Issue II: Selective manipulation of the garbled tables. Recall from the protocol overview of [113] that each row of a garbled table consists of n ciphertexts, the ith one decrypting to a key on the output wire contributed by P_i. During decoding, this decrypted key enables P_i to deduce the masked bit on the output wire. An $\mathcal{A}$ can break unanimity of the protocol of [113] by tweaking the ith ciphertext alone in all the rows for a gate (say with output wire 'c') for some i so that P_i's decrypted key $\overline{k}_c^j$ from ith ciphertext does not match with either key of the pair $(k_{c,0}^i, k_{c,1}^i)$. Now P_i cannot deduce the masked output bit δ_c, while all other honest parties can. This does not disrupt unanimity for the case when c is *not* an output wire of the circuit. Because the incorrect $\overline{k}_c^j$ received by all parties would be used to unmask each of the n ciphertexts of the row corresponding to the gate h where c is an input wire. The decryption would lead to arbitrary values of keys corresponding to all parties for the output wire of h. Since these arbitrary values would not match to the key pairs for all the parties, all honest parties would abort; preserving unanimity.

However, this would be a problem in the case of output gates i.e if c was an output wire of the circuit. To handle this issue, every P_i is additionally made to broadcast its respective pair of keys $(k_{w,0}^i, k_{w,1}^i)$, as a part of output translation table along with their share of mask bits λ_w^i, just for the output wires in Round 4. While processing the output gate, an honest party P_i would not only compare the key obtained upon decryption of the ith ciphertext with its pair of keys, but checks all the keys corresponding to all the ciphertexts with the keys broadcast in Round 4. P_i outputs non-$\perp$ only if all the keys are consistent with a common δ_c. We point that there is no privacy breach since both keys of an honest party is accessible to $\mathcal{A}$ only for the output wires. Finally, we also comment that a rushing $\mathcal{A}$ who now knows the pair of output keys belonging to honest parties can manipulate the ciphertext in such a manner that it decrypts to the flipped value i.e say $k_{c,\overline{\delta_c}}^i$ instead of k_{c,δ_c}^i for all $i \in [n]$. While this would lead to honest parties deducing the wrong value of δ_c and thus potentially a wrong output, this kind of manipulation of output is already taken care by authentication of the output with public verifiability as detailed in the previous issue.

Lastly, we point that in order to preserve unanimity in scenarios where a corrupt party P_i uses the correct witness in ZAP_{ij} but not in ZAP_{ik}; the honest parties check all pairwise ZAP

proofs (facilitated by public-coin property of ZAPs) and abort if any of them fail.

5.7.1.2 Recalling the protocol of [113]

We next present BMR Encoding, Decoding and the back-bone protocol of [113] in Figure 5.14 and Figure 5.15 respectively.

BMR.Encode & BMR.Decode

BMR.Encode

Notations: A Boolean circuit C with W as the number of wires and G as the number of NAND gates (w.l.o.g, assume C to consist of only NAND gates). Let PRF be a pseudo-random function with 4κ-bit output size.

Input: Each party P_i chooses randomness $R_i = \{\lambda_w^i, k_{w,0}^i, k_{w,1}^i, m_{w,0}^i, m_{w,1}^i, h_{w,0}^i, h_{w,1}^i\}_{w\in[W]}$ where λ_w^i is the bit contribution of P_i for the mask of wire w, $(k_{w,0}^i, k_{w,1}^i)$ is the κ-bit PRF key-pair contributed by P_i for wire w, $(m_{w,0}^i, m_{w,1}^i)$ is the 4κ-bit mask-pair contributed by P_i for the key-pair $(k_{w,0}^i, k_{w,1}^i)$ of wire w, and $h_{w,b}^i$ is a hash function from a pairwise-independent family from 4κ to κ bits.

Output: The mask bit for a wire w is computed as: $\lambda_w = \lambda_w^1 \oplus \cdots \oplus \lambda_w^n$ if w is not an input wire, else $\lambda_w = \lambda_w^j$ where w is P_j's input wire. Following are the outputs for $j \in [n], w \in [W], g \in [G]$ such that a, b and c are the input and output wires respectively for gate g:

- Garbled tables: $(C_{\alpha,\beta}^{g,j})_{\alpha,\beta\in\{0,1\}}$, with the ciphertext $C_{\alpha,\beta}^{g,j}$ hiding the mask $m_{\alpha,\beta}^{g,j}$ corresponding to the correct output key $k_{w,b_{\alpha,\beta}}^j$, instead of the key itself. $m_{\alpha,\beta}^{g,j}$ and $C_{\alpha,\beta}^{g,j}$ are computed as:
  $b_{\alpha,\beta}^g = \mathsf{NAND}(\lambda_a \oplus \alpha, \lambda_b \oplus \beta) \oplus \lambda_c = [(\lambda_a \oplus \alpha)\dot{(}\lambda_b \oplus \beta) \oplus 1] \oplus \lambda_c$
  $C_{\alpha\beta}^{g,j} = \left(\bigoplus_{i\in[n]} \mathsf{PRF}_{k_{a,\alpha}^i}^i(g,j,\alpha,\beta)\right) \oplus \left(\bigoplus_{i\in[n]} \mathsf{PRF}_{k_{b,\beta}^i}(g,j,\alpha,\beta)\right) \oplus \left(m_{c,0}^j \oplus b_{\alpha,\beta}^g \dot{(} m_{c,0}^j \oplus m_{c,1}^j)\right)$ (note that, this value is represented as degree-3 monomial)
- Masked keys: $(h_{w,b}^j, \tau_{w,b}^j = h_{w,b}^j(m_{w,b}^j) \oplus k_{w,b}^j)_{b\in\{0,1\}}$
- Keys and masks for input wires w: $\delta_w = \lambda_w \oplus x_w, k_{w,\delta_w}^1 \ldots k_{w,\delta_w}^n$
- Output translation table for output wires w: λ_w

BMR.Decode

Input: Garbled table $C_{\star,\star}^{\star,\star}$, keys k_{w,δ_w}^j for every input wire w and output translation table λ_w.

Computation: For gate g (obtained according to topological ordering) with input wires a, b and output wire c, each P_i computes for $j \in [n]$: $m_c^j = C^{g,j}_{\delta_a,\delta_b} \oplus \bigoplus_{i\in[n]} \left(\mathsf{PRF}_{k^i_{a,\delta_a}}(g, j, \delta_a, \delta_b) \oplus \mathsf{PRF}_{k^i_{b,\delta_b}}(g, j, \delta_a, \delta_b)\right)$. Let δ_c be the bit for which $m_c^i = m^i_{c,\delta_c}$. Set $k^j_{c,\delta_c} := \tau^j_{c,\delta_c} \oplus h^j_{c,\delta_c}(m_c^j)$.

Output: After obtaining δ_w for every output wire w, compute the output value as $\delta_w \oplus \lambda_w$

Figure 5.14: BMR Encoding and Decoding of [113]

Protocol $\pi_{\mathsf{backbone.sa}}$

Inputs: Party P_i has x_i for $i \in [n]$.

Output: $y = f(x_1, \ldots, x_n)$ or $\perp$

Primitives: AMD code $(\mathsf{Encode}, \mathsf{Decode})$, Information-theoretic MAC MAC

Subprotocol: 3-round protocol π_{3bitmult} securely computing any degree-3 polynomial against "defensible" adversary (i.e adversary volunteers a defense (explanation) of its actions until the end of Round 3)

Preprocessing in the start of Round 1: Each P_i does the following -

- Chooses a random MAC key K_i and sets $x'_i = \mathsf{Encode}(x_i, \mathsf{K}_i)$.
- Choose randomness R_i for BMR encoding as per Figure 5.14 for a circuit C defined as follows. Let C' be the circuit that takes input (x_i, K_i) from every party P_i and returns $y = f(x_1, \ldots, x_n)$ and MACs $(t_1 \ldots t_n)$ for y with respect to K_i to every P_i. Then C is the AMD-transformed version of C' that takes AMD-encoding of the input of C' and returns AMD-encoding of the output of C'. Let Set_i denote the set of 3-degree monomials to be computed as a part of the BMR encoding. These monomials constitute the ciphertexts $C^{g,j}_{\alpha,\beta}$ as per Figure 5.14.

Rounds 1-3:

- Run π_{3bitmult} to obtain XOR shares of the monomials in Set_i.
- Each P_i broadcasts $\delta_w = \lambda_w \oplus x_w$ where w is an input wire that belongs to P_i. Note that for input wires, the party that owns the wire chooses the entire λ_w.

Round 4: Each party P_i broadcasts its part of the output of $\mathsf{BMR.Encode}$ in Round 4 as follows:

- Share of Garbled tables: P_i's share of $c^{g,j}_{\alpha,\beta}$ for all gate $g \in [G]$, rows $(\alpha, \beta) \in \{0,1\}^2$ and $j \in [n]$.
- Masked key values for all its key contributions: $\{h^i_{w,b}, \tau^i_{w,b}\}_{b\in\{0,1\}, w\in[W]}$
- Keys for all its input wires w: k^j_{w,δ_w}

- Share of output translation table for output wires w: λ_w^i

Output Computation: P_i computes the output as follows: reconstruct the garbled table and output translation table by XORing the shares obtained in Round 4 and run BMR Decoding Algorithm (Figure 5.14) to obtain AMD-encoded output Y. Obtain the output $(y, t_1 \dots t_n)$ after applying AMD decoding Decode on Y. Output y if t_i validates y as per key K_i, else $\perp$.

Figure 5.15: The back-bone [113] protocol

5.7.1.3 Protocol achieving ua

We present the final protocol with unanimous abort in two steps. First, we modify the foundation protocol $\pi_{\mathsf{backbone.sa}}$ to $\pi_{\mathsf{backbone.ua}}$ in Figure 5.16 to reflect the changes needed to tackle the issues arising from BMR encoding and decoding. Next we attach the ZAPs and related primitives as in [113].

Protocol $\pi_{\mathsf{backbone.ua}}$

Inputs: Party P_i has x_i for $i \in [n]$.

Output: $y = f(x_1, \dots, x_n)$ or $\perp$

Primitives: AMD code $(\mathsf{Encode}, \mathsf{Decode})$, Authentication with Public Verifiability

Subprotocol: Same as in $\pi_{\mathsf{backbone.sa}}$.
Preprocessing: Each P_i does the following:

- Chooses two random secret points K_i, y_i and sets $x_i' = \mathsf{Encode}(x_i, \mathsf{K}_i, y_i)$.
- Choose randomness R_i for BMR encoding as per Figure 5.14 for a circuit C defined as follows: let C' be the circuit that takes input (x_i, K_i, y_i) from every party P_i and returns $y = f(x_1, \dots, x_n)$ and n-degree MAC polynomial $a(x)$ with $a(0) = y$ and $a(\mathsf{K}_i) = y_i$ with respect to verification information (K_i, y_i) chosen by every P_i. Then C is the AMD-transformed version of C' that takes AMD-encoding of the input of C' and returns AMD-encoding of the output of C'.

Rounds 1-3: Same as $\pi_{\mathsf{backbone.sa}}$.

Round 4: Same as $\pi_{\mathsf{backbone.sa}}$. In addition, every P_i broadcasts $(k_{w,0}^i, k_{w,1}^i)$ as a share of output translation table for every output wire w.

Output Computation: P_i computes the output as follows: reconstruct the garbled table and output translation table by XORing the shares obtained in Round 4 and run BMR Decoding Algorithm

(Figure 5.14) to obtain AMD-encoded output Y. Obtain the output $(y^*, a^*(x))$ after applying AMD decoding Decode on Y. Each $P_i (i \in [n])$ outputs $\perp$ if any of the following is true:

- If $a^*(\mathsf{K}_i) \neq y_i$.
- If there exist pairs $a, b \in [n]$ such that $\delta_w^a \neq \delta_w^b$, where δ_w^a (similarly b) be the bit for which the key obtained after decrypting (and subsequently unmasking) the ath (similarly bth) ciphertext i.e. k_w^a matches with $k_{w,\delta_w^a}^a$ (similarly k_w^b matches with $k_{w,\delta_w^b}^b$). This check is done for every output wire w.

Figure 5.16: The back-bone protocol for MPC with ua

The intuition for using the ZAPs in [113] is given below. We emphasize that we retain these proofs in their original form and just recall from [113] for comprehensiveness. The foundation of their actively secure protocol, namely $\pi_{\mathsf{backbone.sa}}$, is secure against a "defensible" adversary which uses a 3-bit multiplication protocol π_{3bitmult} to compute BMR Encoded garbled tables. To keep the attacks by malicious adversary in check, the following tools are used: (1) A 3-round weak one-many non-malleable commitment scheme, $\mathsf{nmcom} = (\mathsf{nmcom}[1], \mathsf{nmcom}[2], \mathsf{nmcom}[3])$ ([111]). This is used to commit to the parties' inputs and randomness in $\pi_{\mathsf{backbone.sa}}$. (2) A 2-round resettable reusable witness indistinguishable proof, $\mathsf{ZAP} = (\mathsf{ZAP}[1], \mathsf{ZAP}[2])$ ([81]). This is used to prove the "correct behaviour" by parties in $\pi_{\mathsf{backbone.sa}}$ so that the attacks by a malicious adversary can be essentially narrowed down to what a defensible adversary can do. In more detail, the first set of ZAPs, ZAP_{ij}^1 is run between each party pair (P_i and P_j) in the first two rounds to prove the correctness of the parties' actions in Round 1 of $\pi_{\mathsf{backbone.sa}}$; and the second set of ZAPs, ZAP_{ij}^2 is run to prove that nmcom (run in Rounds 1-3) commits to a valid witness i.e. input and randomness conforming to the parties' actions in Rounds 1-3 of $\pi_{\mathsf{backbone.sa}}$. Once both the ZAP proofs verify for a particular party (which translates to the adversay having given a valid "defense" at the end of Round 3 of $\pi_{\mathsf{backbone.sa}}$), it can send the shares of the BMR encoding, the masked input keys and the output translation tables in Round 4 to enable BMR decoding and hence, computation of the output.

The modified protocol π_{ua} which provides security with ua uses $\pi_{\mathsf{backbone.ua}}$ as the foundation protocol. The additional primitives of nmcom and ZAPs strapped to $\pi_{\mathsf{backbone.sa}}$ in [113] to achieve security against malicious adversaries are appended to $\pi_{\mathsf{backbone.ua}}$ in the exact same way with one extra `abort` condition: party P_j aborts in Round 3 if any pairwise ZAP ZAP_{jk}^1 or ZAP_{jk}^2 $(j, k \in [n])$ fails. The formal description of the modified protocol appears in Figure 5.17.

Protocol π_{ua}

Inputs: Party P_i has x_i for $i \in [n]$.

Output: $y = f(x_1, \ldots, x_n)$ or $\perp$

Primitives: 3-round non-malleable commitment scheme nmcom with round-wise messages ($\mathsf{nmcom}[1], \mathsf{nmcom}[2], \mathsf{nmcom}[3]$), 2-round resettable reusable witness indistinguishable proof ZAP with round-wise messages ($\mathsf{ZAP}[1], \mathsf{ZAP}[2]$), AMD code ($\mathsf{Encode}, \mathsf{Decode}$).

Subprotocol: $\pi_{\mathsf{backbone.ua}}$ (Figure 5.16).

Preprocessing: Same as protocol $\pi_{\mathsf{backbone.ua}}$.

Round 1: Each P_i ($i \in [n]$) does the following steps:

- Run Round 1 of $\pi_{\mathsf{backbone.ua}}$.
- Engage in two instances of nmcom– nmcom^0_{ij} and nmcom^1_{ij} with every other party P_j, committing to arbitrarily chosen values $w_{0,i}, w_{1,i}$. Let $\mathsf{nmcom}^0_{ij}[1], \mathsf{nmcom}^1_{ij}[1]$ denote the corresponding messages.
- Engage in an instance of ZAP– ZAP^1_{ij} with every other party P_j by sending $\mathsf{ZAP}^1_{ij}[1]$.

Round 2: Each P_i ($i \in [n]$) does the following steps:

- Run Round 2 of $\pi_{\mathsf{backbone.ua}}$.
- Send Round 2 messages of nmcom instances, namely $\mathsf{nmcom}^0_{ij}[2], \mathsf{nmcom}^1_{ij}[2]$.
- Engage in an instance of ZAP– ZAP^2_{ij} with every other party P_j by sending $\mathsf{ZAP}^2_{ij}[1]$.
- Send Round 2 messages of ZAP^1_{ij}, namely $\mathsf{ZAP}^1_{ij}[2]$ to prove correctness of actions in Round 1 of π_{3bitmult}.

Round 3: Each P_i ($i \in [n]$) does the following steps:

- Run Round 3 of $\pi_{\mathsf{backbone.ua}}$.
- Send Round 3 messages of nmcom instances, namely $\mathsf{nmcom}^0_{ij}[3], \mathsf{nmcom}^1_{ij}[3]$.
- Choose $\widetilde{w_{0,i}}, \widetilde{w_{1,i}}$ such that $\widetilde{w_{0,i}} + w_{0,i} = \widetilde{w_{1,i}} + w_{1,i} = \mathsf{wit}_i$ where wit_i is the witness corresponding to the proof of correctness of P_i's actions during π_{3bitmult} with respect to all monomials in Set_i and one instance of nmcom (nmcom^0_{ij} or nmcom^1_{ij} for each j). Broadcast $\widetilde{w_{0,i}}, \widetilde{w_{1,i}}$.

- Send Round 2 message of ZAP^2_{ij}, namely $\mathsf{ZAP}^2_{ij}[2]$ to prove that at least one of $\mathsf{nmcom}^0_{i,j}$ or $\mathsf{nmcom}^1_{i,j}$ is a valid commitment to a valid witness. Namely, for some $b \in \{0,1\}$, $\mathsf{nmcom}^b_{i,j}$ is a valid commitment to $w_{b,i}$ such that $w_{b,i} + \widetilde{w_{b,i}}$ is a valid witness proving correctness of actions of P_i.
- Abort if any pairwise ZAP fails. Public verifiability of the ZAPs enables everyone to agree on this.

Round 4: Each P_i ($i \in [n]$) does the following steps:

- Run Round 4 of $\pi_{\mathsf{backbone.ua}}$.

Output Computation: Same as $\pi_{\mathsf{backbone.ua}}$.

Figure 5.17: Modified Protocol of [113]

5.7.2 Boosting security of [15, 60] to ua

We begin a high-level overview of the compiler presented in the work of [15] which can be used for "compiling" any 3-round semi-malicious MPC protocol (with first round being public-coin) into a 4-round MPC protocol achieving sa against dishonest majority. The primary tools used in the compiler are a non-interactive commitment NCom, three-message delayed-input distributional weak zero-knowledge argument system WZK, three-message delayed-input extractable commitment scheme Ecom, three-message trapdoor generation protocol TDGen, three-message delayed-input witness-indistinguishable argument system WI, a three round delayed-input witness-indistinguishable argument with non-adaptive bounded rewinding security RWI and three-message non-malleable commitment scheme NMComn.

The skeleton of the 4-round protocol π_{mal} compiling the underlying 3-round semi-malicious protocol, say π_{sm}, is as follows: The rounds 1, 2 and 3 of π_{sm} are run during Rounds 1, 3 and 4 of π_{mal} respectively. Each party P_i participates in the 3-round subprotocols Ecom, NMComn and TDGen in Round 1 - 3 of π_{mal}; where Ecom and NMComn are used to compute commitments on $(\mathsf{x}_i, \mathsf{r}_i)$ i.e the input and randomness used in the protocol and $\perp$ respectively. In parallel, each P_i computes a non-interactive commitment nc_i to value 1 using NCom and proves via WZK run in Rounds 1 - 3 that nc_i is indeed a commitment to 1. Furthermore, RWI, run in Rounds 1 - 3 between every pair of parties, is used by each P_i (prover) to prove towards verifier P_j ($j \neq i$) that Round 2 message of π_{sm} (sent during Round 3 of π_{mal}) was honestly computed based on $(\mathsf{x}_i, \mathsf{r}_i)$ committed in its instance of Ecom and the Round 1 transcript of π_{sm}. The alternative statements for RWI that are used for simulation purpose include commitment to valid trapdoor using NMCom and nc being a commitment to 0. Lastly, the 3-round WI, run in Rounds 1, 2 and

4 between every pair of parties, is used as means for each P_i (prover) to prove towards verifier P_j ($j \neq i$) that Round 3 message of π_{sm} (sent during Round 4 of π_{mal}) was honestly computed based on $(\mathsf{x}_i, \mathsf{r}_i)$ committed in its instance of Ecom and the Round 2 transcript of π_{sm}. The alternative statement for WI used for simulation includes commitment to valid trapdoor using NMCom. This completes the high-level overview of the compiler focusing on just the relevant details.

The above described 4-round protocol π_{mal} achieves only security with selective abort as the $\mathsf{RWI}, \mathsf{WI}$ and WZK proofs are executed pairwise and allow a corrupt party to selectively misbehave to a subset of honest parties; thereby keeping them on different pages. To boost its security to ua, we propose the following modifications: First, if an honest party acting as a verifier in WZK or RWI detects that any of the proofs have failed at the end of Round 3, she broadcasts `abort` in Round 4. If any of the parties broadcast `abort`, all honest parties simply output $\perp$. This tweak would ensure that even private misbehaviour by an adversary upto Round 3 is made public to all by Round 4, enabling unanimity. Finally, in order to maintain unanimity at the end of Round 4, we make all parties check each of the public-coin pairwise witness-indistinguishable instances (WI) (instantiated with [142]) completing in Round 4 and abort if any of them failed (as opposed to only the pairwise WI instances where the party acts as verifier). Thus, the above mentioned modifications incorporated in the protocol of [15] produces a 4-round protocol achieving ua in dishonest majority.

Boosting security to identifiable abort. We observe that the security of the (modified) protocol of [15] can be boosted to identifiable abort upon applying the following tweaks: First, as described above, the actions of the parties are made publicly verifiable by making all parties check each of the pairwise public-coin witness-indistinguishable proofs (as opposed to only the ones where the party acts as verifier). Next, the private misbehavior in the 3-round weak zero-knowledge (WZK) can be made public by allowing the verifier of the WZK to publish the randomness used in the WZK in the last round (after the WZK instance has been completed).

Lastly, we point that the techniques of boosting security of [15] to ua, namely making private misbehaviour upto Round 3 public by broadcasting `abort` in Round 4 and making all the parties check each of the pairwise WI proofs (completing in Round 4) can be used to boost the security of [60] to ua as well.

5.8 Appendix: Towards obtaining a 4-round (god|ua)-BoBW protocol

In this section, we present the sketch of a 4-round (god|ua)-BoBW protocol based on sub-exponentially secure trapdoor permutations and ZAPs. We believe that these preliminary ideas are promising to either prove the impossibility or build a construction of a 4-round (god|ua)-BoBW protocol in the plain model under polynomial-time assumptions.

Firstly, we note that in order to compile our delayed-semi-maliciously secure (god|ua)-BoBW to the malicious setting, the honest parties must unanimously agree on the identity of the parties who have misbehaved till the penultimate round. To achieve the optimal round complexity of 4, this would demand a 3-round publicly verifiable proof that would prove correctness of the actions upto the penultimate round. Thus, the absence of a 3-round zero-knowledge (ZK) proof seems to constitute the primary bottleneck in building a 4-round maliciously-secure (god|ua)-BoBW in the plain model. Since the existing compilers achieving security with abort within 4 rounds based on polynomial-time assumptions such as [15, 60] (which rely on weakened notion of zero-knowledge, namely promise ZK) do not have the feature of public verifiability at the end of Round 3, we build upon the compiler of [64] based on sub-exponentially secure trapdoor permutations and ZAPs, which offers this property.

The structure of the compiler of [64] that compiles a 3-round delayed semi-malicious protocol, say π_{dsm} to a 4-round malicious protocol, say π_{mal} is as follows: Each party commits to her input in Round 1 of π_{mal} using a non-interactive commitment scheme. The 3 rounds of π_{dsm} are executed in Rounds 2- 4 of π_{mal}. To prove correctness of first two rounds of π_{dsm}, the parties commit to their randomness and input (which represent a defence for π_{dsm}) using a special non-malleable commitment scheme (satisfying additional properties of honest-extractable, delayed-input, reusable decommitment information and last-message psuedorandomness; refer [64] for details), and prove via ZAP (in Rounds 2 - 3) that this commitment actually contains a valid defence. Next, the parties engage in a 4-round delayed-input Non-Malleable Zero-Knowledge (NMZK) argument to prove correctness of Round 3 of π_{dsm} (wrt the defence committed in the non-malleable commitment scheme and the non-interactive commitment). There are two additional components to aid the simulator– First, a 3-round witness-indistinguishable proof of knowledge (WIPoK) between every pair of parties where each party proves to the other the knowledge of a secret information (specifically knowledge of a value y such that $f(y) = Y_0$ or $f(y) = Y_1$, given that f is a one-way permutation where (Y_0, Y_1) is chosen by the prover). Second, another special non-malleable commitment of a random string. To be more specific, simulator acting on behalf of honest P_i extracts the trapdoor (the preimage y of the OWP)

from the WIPoK instance with corrupt P_j as the prover. Next, the simulator commits to this trapdoor using the special non-malleable commitment scheme, which will be used as witness for the ZAP (with P_i as prover and corrupt P_j as verifier). This completes the high-level description of the protocol.

To construct the 4-round (god|ua)-BoBW in the plain model, we plug in our 3-round delayed semi-malicious BoBW protocol in the above compiler. Similar to our modifications over the compiler of [35], parties are made to set the boolean indicators flag_i to 0 if malicious behavior of P_i is detected in the first three rounds. It is easy to check that all parties agree on the flag values as the components of the compiler upto Round 3 including the ZAP are publicly verifiable. With the above change, the BoBW guarantees of the underlying delayed semi-malicious protocols are translated to the malicious setting as well. To avoid rewinding of messages in the underlying delayed semi-malicious protocol, we run the 3-rounds of our delayed semi-malicious protocol in Round 1, 2 and 4 of the 4-round compiled maliciously secure protocol. This completes the sketch of the 4-round (god|ua)-BoBW protocol in the plain model relying on the assumptions of the compiler of [64], namely sub-exponentially secure trapdoor permutations and ZAPs.

Before concluding this section, we give the sketch of the simulation. As per the simulator of [64], the simulator extracts the trapdoor by rewinding the adversary from the third to the second round (referred to as look-ahead threads). This means that before the rewinds the simulator needs to use a valid witness for the ZAP without knowing the trapdoor. For this purpose, the simulator during the look-ahead rewinding threads uses a valid defence for π_{dsm} with a random input. After the extraction, the simulator rewinds up to the second round, commits to the trapdoor, uses the simulator of the underlying π_{dsm} protocol and completes the ZAP proof using the knowledge of the trapdoor. We use the same simulation strategy for our BoBW protocol as well except for the following change: Unlike the simulator of [64], the simulator of our BoBW protocol cannot halt incase a corrupt party aborts (in order to achieve god in honest majority setting). We thereby follow the simulation strategy as described for our 5-round protocol $\pi_{\mathsf{bw.god.plain}}$ - The simulator proceeds to rewinds and extracts trapdoors and inputs of corrupt parties who are alive (have not aborted upto Round 3). The halting condition of the simulator and the security argument is similar to that of $\pi_{\mathsf{bw.god.plain}}$. This completes the proof sketch.

Chapter 6

On the Round Complexity of Fair and Robust MPC against Dynamic and Boundary Adversaries

In this chapter, we investigate the round complexity of fair and robust (achieving god) MPC against two powerful and generalized adversaries, namely the dynamic and boundary adversaries. Our results in these corruption settings overcome the demarcation of the study of round complexity of MPC based on the adversarial behaviour (either active or passive). Specifically, we extend the study of round complexity of fair and robust MPC beyond the traditional settings of passive majority (where majority of the parties are passively corrupt) and active minority (where minority of the parties are actively corrupt) to include mixed adversaries who can simultaneously perform both active and passive corruptions.

6.1 Introduction

Two of the most sought-after properties of Multi-party Computation (MPC) protocols are fn and god, the latter also referred to as robustness. Achieving both, however, brings in the necessary requirement of malicious-minority. In a generalised adversarial setting where the adversary is allowed to corrupt both actively and passively, the necessary bound for a n-party fair or robust protocol turns out to be $t_a + t_p < n$, where t_a, t_p denote the threshold for active and passive corruption with the latter subsuming the former. Subsuming the malicious-minority as a boundary special case, this setting, denoted as dynamic corruption, opens up a range of possible corruption scenarios for the adversary. While dynamic corruption includes the entire range of thresholds for (t_a, t_p) starting from $(\lceil \frac{n}{2} \rceil - 1, \lfloor \frac{n}{2} \rfloor)$ to $(0, n-1)$, the boundary corruption

restricts the adversary only to the boundary cases of $(\lceil \frac{n}{2} \rceil - 1, \lfloor \frac{n}{2} \rfloor)$ and $(0, n-1)$. Notably, both corruption settings empower an adversary to control majority of the parties, yet ensuring the count on active corruption never goes beyond $\lceil \frac{n}{2} \rceil - 1$.

As detailed in Section 1.4.3 the protocols in dynamic and boundary setting offer strong defence and are more tolerant and better-fit in practical scenarios where the attack can come in many unforeseen ways. We target the round complexity of fair and robust MPC tolerating dynamic and boundary adversaries. While a detailed description of our results appears in Section 1.4.3, we present the related work and a brief summary of the results below.

Related Work. The relevant literature of round complexity of fair and robust MPC protocols in the traditional adversarial settings involving only single type of adversary (either passive or active) is outlined in Section 1.3. Moving on to the setting of generalized adversary, there are primarily two adversarial models that are most relevant to us. The first model initiated by [79] consider a mixed adversary (referred to as graceful degradation of *corruptions*) that can *simultaneously* perform different types of corruptions. Feasibility results in this model appeared in the works of [86, 87, 119, 22]. The dynamic-admissible adversary considered in our work is consistent with this model since it involves simultaneous active and passive corruptions. The second model proposed by [54] concerns protocols that are secure against an adversary that can either choose to corrupt a subset of parties with particular corruption type (say, passively) or alternately a different subset (typically smaller) of parties with a second corruption type (say, actively), but only *single* type of corruption occurs at a time. Referred to as graceful degradation of *security* [54, 152, 88, 89, 124, 134, 127], such protocols achieve different security guarantees based on the set of corrupted parties; for instance robustness/information-theoretic security against the smaller corruption set and abort/computational security against the larger corruption set. We note that the boundary-admissible adversary when n is odd, involves either purely active (since $t_a = t_p$ holds when $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$) corruptions or purely passive corruptions (where $(t_a, t_p) = (0, n-1)$); thereby fitting in the second model (Infact, boundary-admissible adversary for odd n degenerates to the adversarial model studied in "best-of-both-worlds" MPC [127]). However, in case of even n, the boundary-admissible adversary with $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ would involve simultaneous passive and active corruption as $t_p = t_a + 1$ and fit in the prior model. Lastly, both graceful degradation of security and corruptions were generalized in the works of [120, 122]. To the best of our knowledge, the interesting and natural question of round complexity has not been studied in these stronger adversarial models.

6.1.1 Our Results

We settle the question of exact round complexity of fair and robust MPC tolerating dynamic and boundary adversaries in the public and private setup models. As it turns out, $\lceil \frac{n}{2} \rceil + 1$ rounds are necessary and sufficient for fair as well as robust MPC tolerating dynamic corruption. The non-constant barrier raised by dynamic corruption can be sailed through for a boundary adversary. The round complexity of 3 and 4 is necessary and sufficient for fair and GOD protocols respectively, with the latter having an exception of allowing 3 round protocols in the presence of a single active corruption. While all our lower bounds assume pair-wise private and broadcast channels and are resilient to the presence of both public (CRS) and private (PKI) setup, our upper bounds are broadcast-only and assume only public setup. The traditional and popular setting of malicious-minority, being restricted compared to both dynamic and boundary setting, requires 3 and 2 rounds in the presence of public and private setup respectively for both fair as well as GOD protocols. The need for CRS in our constructions stems from the underlying 2-round protocol achieving unanimous or identifiable abort. We leave open the question of constructing tight upper bounds or coming up with new lower bounds in the plain model.

Adversary	Security	Rounds	Lower bound	Upper Bound
Passive-majority	fn, god	2	[112] (private)	[93, 35] (plain)
Malicious-minority	fn, god	3	[108, 166] (public)	[4, 16] (plain)
	fn, god	2	[112] (private)	[108] (private)
Boundary	fn	**3**	**Our Work** [169] (private)	**Our Work** [169] (public)
	god	**4** (**3** when $t_a \leq 1$)	**Our Work** [169] (private)	**Our Work** [169] (public)
Dynamic	fn, god	$\lceil \frac{\mathbf{n}}{\mathbf{2}} \rceil + \mathbf{1}$	**Our Work** [169] (private)	**Our Work** [169] (public)

6.1.2 Techniques

In this section, we give a glimpse into the techniques used in our lower bounds and matching upper bound constructions.

Lower Bounds. We present 3 lower bounds, all of which hold assuming access to *both* CRS and PKI– **(a)** $\lceil n/2 \rceil + 1$ rounds are necessary to achieve fairness against dynamic adversary. **(b)** 4 rounds are necessary to achieve robustness against a boundary adversary. **(c)** 3 rounds are necessary to achieve fairness against a boundary adversary.

The first lower bound **(a)** effectively captures the power of dynamic corruption stemming from the ambiguity caused by the total range of thresholds (t_a, t_p) starting from $(\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ to $(0, n-1)$. The proof navigates through this sequence starting with maximal active corruption and proceeds to scenarios of lesser active corruptions one at a time. An inductive

argument neatly captures how the value of t_p growing alongside decreasing values of t_a can be exploited by adversarial strategies violating fairness, eventually dragging the round complexity all the way upto $\lceil n/2 \rceil + 1$. The lower bounds **(b)** and **(c)** are shown by considering a specific set of small number of parties and assume the existence of a 3 (2) round robust (fair) protocol for contradiction respectively. Subsequently, inferences are drawn based on cleverly-designed strategies exploiting the properties of GOD and fairness. These inferences and strategies are interconnected in a manner that builds up to a strategy violating privacy, thereby leading to a final contradiction.

Upper Bounds. We present 5 upper bounds, in the broadcast-only setting comprising of two upper bounds each for fairness and GOD against dynamic and boundary adversary respectively and lastly, an additional 3-round upper bound for GOD against the special case of single malicious corruption by boundary adversary in order to demonstrate the circumvention of lower bound **(b)**. Tightness of this upper bound follows from lower bound **(c)** (that holds for single malicious corruption) as GOD implies fairness. Our upper bounds can be viewed as "compiled" protocols obtained upon plugging in any 2-round broadcast-only protocols [93, 35] achieving unanimous abort against malicious majority. While the fair upper-bounds do not require any additional property from the underlying 2-round protocol, our robust protocols demand the property of *identifiable abort* and *function-delayed* property i.e the first round of the protocol is independent of the function to be computed and the number of parties. Looking ahead, this enables us to run many parallel instances of the round 1 in the beginning and run the second round sequentially as and when failure happens to compute a new function (that gets determined based on the identities of the corrupt parties). Assumption wise, all our upper bound constructions rely on 2-round maliciously-secure oblivious transfer (OT) in common random/reference string models. We now give a high-level overview of the specific challenges we encounter in each of our upper bounds and the techniques we use to tackle them.

Dynamic adversary: The two upper bounds against dynamic adversary show sufficiency of $\lceil n/2 \rceil + 1$ rounds to achieve fairness and robustness against dynamic admissible adversary. The upper bound for fairness is built upon the protocol of [122] that introduces a special-kind of sharing, which we refer to as levelled-sharing where a value is divided into summands (adding upto the value) and each summand is shared with varying degrees. The heart of the protocol of [122] lies in its gradual reconstruction of the levelled-shared output (obtained by running an MPC protocol with unanimous abort), starting with the summand corresponding to the highest degree down to the lowest. The argument for fairness banks on the fact that the more the adversary raises its disruptive power in an attempt to control reconstruction of more number of summands, the more it looses its eavesdropping capability and consequently

learns fewer number of summands by itself and vice versa. This discourages an adversary from misbehaving as using maximal disruptive power reduces its eavesdropping capability such that he falls short of learning the next summand in sequence without the help of honest parties. The innovation of our fair protocol lies in delicately fixing the parameters of levelled-sharing in a manner that optimal round complexity can be attained whilst maintaining fairness.

Next, we point that since the fair protocol consumes the optimal round complexity of $\lceil n/2 \rceil + 1$ even in the case of honest execution, the primary hurdle in our second upper bound is to be able to carry out re-runs when an adversary disrupts computation to achieve robustness without consuming extra rounds. Banking on the player-elimination technique, we use identifiability to bar the corrupt parties disrupting computation from participating thereafter. Having parallel execution of Round 1 of all the required re-reruns helps us get closer to the optimal bound. While these approaches aid to a great extent, the final saviour comes in the form of a delicate and crucial observation regarding how the thresholds of the levelled-sharing can be manipulated carefully, accounting for the cheaters identified so far. This trick exploits the pattern of reduced corruption scenarios obtained upon cheater identification and helps to compensate for the rounds consumed in subprotocols that were eventually disrupted by the adversary. The analysis of the round complexity of the protocol being subtle, we use an intricate recursive argument to capture all scenarios and show that the optimal lower bound is never exceeded. Lastly, we point that both upper bound constructions against dynamic adversary assume equivocal non-interactive commitment (such as Pedersen commitment [175]). The GOD upper bound additionally assumes the existence of Non-Interactive Zero-Knowledge (NIZK) in the common random/reference string model.

Boundary adversary: The three upper bounds against boundary-admissible adversary restricted to corruption scenarios either $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ or $(t_a, t_p) = (0, n-1)$ show that **(a)** 4 rounds are sufficient to achieve robustness against boundary-admissible adversary **(b)** 3 rounds are sufficient to achieve robustness against special-case boundary-admissible adversary when $t_a \leq 1$ i.e adversary corrupts with parameters either $(t_a, t_p) = (1, \lfloor n/2 \rfloor)$ or $(t_a, t_p) = (0, n-1)$ **(c)** 3 rounds are sufficient to achieve fairness against boundary-admissible adversary. At a high-level, all the three upper bounds begin with a 2-round protocol secure against malicious majority that computes threshold sharing of the output. Intuitively, this seems to serve as the only available option as protocols customized for malicious minority typically breach privacy when views of majority of the parties are combined (thereby will break down against $t_p < n$ semi-honest corruptions). On the flip side, protocols customized for exclusively passive majority may violate correctness/privacy in the presence of even single malicious corruption. Subsequently, this natural route bifurcates into two scenarios based on whether

the adversary allows the computation of the threshold sharing of output to succeed or not. In case of success, all the three upper bounds proceed via the common route of reconstruction which is guaranteed to be robust by the property of threshold sharing. The distinctness of the 3 settings (accordingly the upper bounds) crops up in the alternate scenario i.e. when the computation of threshold sharing of output aborts. While in upper bound **(c)**, parties simply terminate with $\perp$ maintaining fairness enabled by privacy of the threshold sharing; the upper bounds **(a)** and **(b)** demanding stronger guarantee of robustness cannot afford to do so. These two upper bounds exploit the fact that the corruption scenario has now been identified to be the boundary case having active corruptions, thereby protocols tolerating malicious minority can now be executed. While the above outline is inspired by the work of [127], we point that we need to tackle the exact corruption scenarios as that of the protocols of [127] only when n is odd. On the other hand when n is even, the extreme case for active corruption accommodates an additional passive corruption ($t_p = t_a + 1$). Apart from hitting the optimal round complexity, tackling the distinct boundary cases for odd and even n in a unified way brings challenge for our protocol. To overcome these challenges, in addition to techniques of identification and elimination of corrupt parties who disrupt computation, we employ tricks such as parallelizing without compromising on security to achieve the optimum round complexity. Assumption wise, while both the robust constructions **(a)** and **(b)** rely on NIZKs, the former additionally assumes Zaps (2-round, public-coin witness-indistinguishable protocols) and public-key encryption.

Lastly, we present the model and useful definitions below before proceeding to the technical sections.

Model and Definitions. We consider a set of PPT parties $\mathcal{P} = \{P_1, \dots P_n\}$. Our upper bounds assume the parties connected by a broadcast channel and a setup where parties have access to common reference string (CRS). Our lower bounds hold even when the parties are additionally connected by pairwise-secure and authentic channels and for a stronger setup, namely assuming access to CRS as well as public-key infrastructure (PKI). We assume that there exists a PPT adversary $\mathcal{A}$, who can corrupt a subset of these parties.

We consider two kinds of adversarial settings in this work. In both settings, the $\mathcal{A}$ is characterised by two thresholds (t_a, t_p), where he may corrupt upto t_p parties passively, and upto t_a of these parties even actively. Note that t_p is the total number of passive corruptions that includes the active corruptions and additional parties that are exclusively passively corrupt. We now define dynamic and boundary admissible adversaries.

Definition 6.1 (Dynamic-admissible Adversary) *An adversary attacking an n-party MPC protocol with threshold (t_a, t_p) is called dynamic-admissible as long as $t_a + t_p < n$ and $t_a \leq t_p$.*

For dynamic-admissible adversary, we denote the set of active and passively corrupt parties by $\mathcal{D}$ and $\mathcal{E}$ respectively, where $|\mathcal{D}| = t_a$ and $|\mathcal{E}| = t_p$.

Definition 6.2 (Boundary-admissible Adversary) *An adversary attacking an n-party MPC protocol with threshold (t_a, t_p) is called boundary-admissible as long as he corrupts either with parameters (a) $(t_a, t_p) = (\lceil \frac{n}{2} \rceil - 1, \lfloor n/2 \rfloor)$ or (b) $(t_a, t_p) = (0, n-1)$.*

In our work, we also consider a special-case of boundary adversary with $t_a \leq 1$ where the adversary corrupts either with parameters $(t_a, t_p) = (1, \lfloor n/2 \rfloor)$ or $(t_a, t_p) = (0, n-1)$.

Roadmap. Our lower and upper bounds for dynamic and boundary corruption appear in Sections 6.2-6.3 and in Sections 6.4-6.5 respectively. The security definition and the functionalities appear in Chapter 2.

6.2 Lower Bounds for Dynamic Corruption

In this section, we show that $\lceil \frac{n}{2} \rceil + 1$ rounds are necessary to achieve MPC with fairness against a dynamic-admissible $\mathcal{A}$ with threshold (t_a, t_p). This result shows impossibility of constant-round fair and robust protocols in the setting of dynamic corruption.

Theorem 6.1 *No $\lceil \frac{n}{2} \rceil$-round n-party MPC protocol can achieve fairness tolerating a dynamic-admissible adversary $\mathcal{A}$ with threshold (t_a, t_p) in a setting with pairwise-private and broadcast channels, and a setup that includes CRS and PKI.*

Proof: We prove the theorem by contradiction. Suppose there exists a $\lceil \frac{n}{2} \rceil$-round n-party MPC protocol π computing any function $f(x_1 \dots x_n)$ (where x_i denotes the input of party P_i) that achieves fairness against a dynamic-admissible $\mathcal{A}$ with corruption threshold (t_a, t_p) and in the presence of a setup with CRS and PKI. At a high-level, our proof argument defines a sequence of hybrid executions of π, navigating through all the possible admissible corruption scenarios assuming $t_a + t_p = n - 1$ and starting with the maximum admissible value of $t_a = \lceil n/2 \rceil - 1$. Our first hybrid under the spell of a dynamic-admissible adversary, corrupting $\lceil n/2 \rceil - 1$ parties actively and stopping their communication in the last round, lets us conclude that the joint view of the honest and passively-corrupted parties by the end of penultimate round must hold the output in order for π to satisfy fairness. If not, while ceasing communication in the last round does not prevent $\mathcal{A}$ from getting all the messages in the last round and thereby the output, the honest parties do fail to compute the output due to the non-cooperation of t_a parties, violating fairness. The views of the passively corrupt parties need to be taken into account as they follow protocol steps correctly and assist in output computation.

Leveraging the fact that drop of t_a leads to rise of t_p, we then propose a new hybrid where t_a is demoted by 1 and consequently t_p grows big enough to subsume the list of honest and passive-corruption from the previous hybrid. As the view of the adversary in this hybrid holds the output by the end of penultimate round itself, its actively-corrupt parties need not speak in the penultimate round. Now fairness in the face of current strategy of the actively-corrupted parties needs the joint view of the honest and passively-corrupted parties by the end of $\lceil n/2 \rceil - 2$ round to hold the output. This continues with the set of honest and passively-corrupted parties growing by size one between every two hybrids. Propagating this pattern to the earlier rounds eventually lets us conclude that an adversary with threshold $(t_a, t_p) = (0, n-1)$ (no active corruption case) can obtain the output at the end of Round 1 itself. This leads us to a final strategy that violates privacy of π via residual attack. This completes the proof sketch. We now prove the sequence of lemmas to complete the proof.

Lemma 6.1 *In an execution of π where all parties behave honestly upto (and including) Round $(\lceil \frac{n}{2} \rceil - i)$ for $i \in [\lceil \frac{n}{2} \rceil - 1]$, there exists a set of parties S^i with size $(\lfloor \frac{n}{2} \rfloor + i)$ whose combined view at the end of Round $\lceil \frac{n}{2} \rceil - i$ suffices to compute the output, with overwhelming probability.*

Proof: We prove the lemma by induction. Let $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ denote the set of parties and $\mathcal{D}(\mathcal{E})$ denote the set of actively (passively) corrupt parties where $\mathcal{D} \subseteq \mathcal{E}$. Here $|\mathcal{D}| = t_a$ and $|\mathcal{E}| = t_p$.

Base Case ($i = 1$): We consider an execution of the protocol π with a dynamic-admissible adversary $\mathcal{A}$ corrupting parties with threshold $(t_a, t_p) = (\lceil \frac{n}{2} \rceil - 1, \lfloor n/2 \rfloor)$ and an adversarial strategy $\mathcal{A}_1$ as follows. The set of actively corrupt parties $\mathcal{D}$ behave honestly upto (and including) Round $\lceil \frac{n}{2} \rceil - 1$ and simply remain silent in the last round i.e the $\lceil \frac{n}{2} \rceil$th round. Since $\mathcal{A}$ receives all the desired communication throughout the protocol, it follows directly from the correctness of π that $\mathcal{A}$ must be able to compute the output with overwhelming probability. Since π is assumed to be fair, the honest parties must also be able to compute the output even without the $\lceil \frac{n}{2} \rceil$th round communication from parties in $\mathcal{D}$. We can now conclude that the combined view of parties in $\mathcal{P} \setminus \mathcal{D}$ at the end of Round $\lceil \frac{n}{2} \rceil - 1$ must suffice to compute the output. Thus, the set $S^1 = \mathcal{P} \setminus \mathcal{D}$ of parties with size $n - t_a = n - (\lceil \frac{n}{2} \rceil - 1) = \lfloor \frac{n}{2} \rfloor + 1$ hold a combined view at the end of Round $\lceil \frac{n}{2} \rceil - 1$ that suffices to compute the output with overwhelming probability. This completes the base case.

Induction Hypothesis ($i = \ell$). Suppose the statement is true for $i = \ell$ i.e. if all parties behave honestly upto (and including) Round $(\lceil \frac{n}{2} \rceil - \ell)$, then there exists a set of parties, say S^ℓ, with $|S^\ell| = (\lfloor \frac{n}{2} \rfloor + \ell)$ whose combined view at the end of $(\lceil \frac{n}{2} \rceil - \ell)$th round, suffices to compute the output, with overwhelming probability.

Induction Step ($i = \ell + 1$)**.** We consider an execution of the protocol π with a dynamic-admissible adversary $\mathcal{A}$ corrupting parties with threshold $(t_a, t_p) = (\lceil \frac{n}{2} \rceil - \ell - 1, \lfloor \frac{n}{2} \rfloor + \ell)$ and $\mathcal{E} = S^{\ell}$ as defined in the induction hypothesis and an adversarial strategy $\mathcal{A}_{\ell+1}$ as follows. The set of actively corrupt parties $\mathcal{D}$ behave honestly upto (and including) Round $(\lceil \frac{n}{2} \rceil - \ell - 1)$ and simply remain silent from Round $(\lceil \frac{n}{2} \rceil - \ell)$ onwards. Since $\mathcal{A}$ receives all the desired communication upto (and including) Round $(\lceil \frac{n}{2} \rceil - \ell)$ of π (as per an honest execution) on behalf of parties in $\mathcal{E}$, it follows directly from the induction hypothesis that the combined view of the parties in $\mathcal{E}$ where $|\mathcal{E}| = \lfloor \frac{n}{2} \rfloor + \ell$ must suffice to compute the output, with overwhelming probability. Since π is assumed to be fair, the honest parties must also be able to compute the output even though the parties in $\mathcal{D}$ stop communicating from Round $(\lceil \frac{n}{2} \rceil - \ell)$ onwards. We can now conclude that the combined view of parties in $\mathcal{P} \setminus \mathcal{D}$ at the end of Round $(\lceil \frac{n}{2} \rceil - \ell - 1)$ must suffice to compute the output. Thus, the set $S^{\ell+1} = \mathcal{P} \setminus \mathcal{D}$ of parties with size $n - t_a = n - (\lceil \frac{n}{2} \rceil - \ell - 1) = \lfloor \frac{n}{2} \rfloor + \ell + 1$ hold a combined view at the end of Round $(\lceil \frac{n}{2} \rceil - \ell - 1)$ that suffices to compute the output with overwhelming probability. This completes the induction hypothesis and the proof of Lemma 6.1. □

Lemma 6.2 *There exists an adversary $\mathcal{A}$ that is able to compute the output at the end of Round 1 of π with overwhelming probability.*

Proof: When $i = \lceil \frac{n}{2} \rceil - 1$, Lemma 6.1 implies that if all parties behave honestly in Round 1, then there exists a set $S^{\lceil \frac{n}{2} \rceil - 1}$ of $(\lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil - 1) = (\lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil - 1) = n - 1$ parties whose combined view suffices to compute the output at the end of Round 1, with overwhelming probability. Consequently, a dynamic-admissible adversary $\mathcal{A}$ corrupting the parties with threshold $(t_a, t_p) = (0, n-1)$ and $(\mathcal{D} = \emptyset, \mathcal{E} = S^{\lceil \frac{n}{2} \rceil - 1})$ must be able to compute the output at the end of Round 1 itself. □

Lemma 6.3 *Protocol π does not achieve privacy.*

Proof: It follows directly from Lemma 6.2 that there exists an adversary $\mathcal{A}$ with threshold $(t_a, t_p) = (0, n-1)$ corrupting a set of $(n-1)$ parties passively, say $\mathcal{E} = \{P_1, \ldots P_{n-1}\}$, that is able to compute the output at the end of Round 1 itself, with overwhelming probability. Thus, $\mathcal{A}$ can obtain multiple evaluations of the function f by locally plugging in different values for $\{x_1, \ldots, x_{n-1}\}$ while honest P_n's input x_n remains fixed. This residual function attack violates privacy of P_n. As a concrete example, let f be a common output function computing $x_1 \wedge x_n$, where x_i ($i \in \{1, n\}$) denotes a single bit. During the execution of π, $\mathcal{A}$ behaves honestly with input $x_1 = 0$ on behalf of P_1. However, the passively-corrupt P_1 can locally plug-in $x_1 = 1$ and learn x_n (via the output $x_1 \wedge x_n$). This is a clear breach of privacy, as in the ideal world,

$\mathcal{A}$ participating honestly with input $x_1 = 0$ on behalf of P_1 would learn nothing about x_n; in contrast to the execution of π where $\mathcal{A}$ learns x_n regardless of his input. This completes the proof. □

We have thus arrived at a contradiction to our assumption that π securely computes f and achieves fairness. This completes the proof of Theorem 6.1.

□

For better understanding, we illustrate the adversarial strategies and implications derived with respect to the specific case of $n = 7$ and 4-round ($\lceil n/2 \rceil = 4$) protocol π in the Table below. The last column (S, r) indicates the implication that the combined view of parties in S $(= \mathcal{P} \setminus \mathcal{D})$ at the end of Round number r suffices to compute the output.

(t_a, t_p)	$\mathcal{D}$	$\mathcal{E}$	Strategy of $\mathcal{A}$	S, r
$(3,3)$	$\{P_1, P_2, P_3\}$	$\{P_1, P_2, P_3\}$	Stop $\mathcal{D}$ after R3	$S^1 = \{P_4, P_5, P_6, P_7\}$, R3
$(2,4)$	$\{P_6, P_7\}$	$\{P_4, P_5, P_6, P_7\}$ (i.e S^1)	Stop $\mathcal{D}$ after R2	$S^2 = \{P_1, P_2, P_3, P_4, P_5\}$, R2
$(1,5)$	$\{P_1\}$	$\{P_1, P_2, P_3, P_4, P_5\}$ (i.e S^2)	Stop $\mathcal{D}$ after R1	$S^3 = \{P_2, P_3, P_4, P_5, P_6, P_7\}$, R1
$(0,6)$	$\emptyset$	$\{P_2, P_3, P_4, P_5, P_6, P_7\}$ (i.e S^3)	Residual attack wrt $\mathcal{E}$	$---$

6.3 Upper bounds for Dynamic Corruption

In this section, we describe two n-party upper bounds tolerating a dynamic-admissible adversary $\mathcal{A}$ with threshold (t_a, t_p). The first upper bound achieves fairness and is a stepping stone to the construction of the second upper bound that achieves guaranteed output delivery. Both the upper bounds comprise of $\lceil n/2 \rceil + 1$ rounds in the presence of CRS, tightly matching our lower bound result of Section 6.2. We start with an important building block needed for both the fair and GOD protocols.

6.3.1 Levelled-sharing of a secret

Our protocols in the dynamic corruption setting involve a special kind of sharing referred as levelled sharing, which is inspired by and a generalized variant of the sharing defined in [122]. The sharing is parameterized with two thresholds, α and β with $\alpha \geq \beta$, that dictate the number of levels as $\alpha - \beta + 1$. To share a secret in (α, β)-levelled-shared fashion, $\alpha - \beta + 1$ additive shares (levels) of the secret, indexed from α to β are created and each additive share is then Shamir-shared [180] using polynomial of degree that is same as its assigned index. Further each Shamir-sharing is authenticated using a non-interactive commitment scheme, to ensure detectably correct reconstruction. For technical reasons in the simulation-based security proof, we need an instantiation of commitment scheme that allows equivocation of commitment to any

message with the help of trapdoor and provides statistical hiding and computational binding. Denoting such a commitment scheme by eNICOM (Equivocal Non-Interactive Commitment), we present both the formal definition and an instantiation based on Pedersen's commitment scheme [175] in Section 2.4.2.1. While the sharing will involve the entire population $\mathcal{P}$ in our fair protocol, it may be restricted to many different subsets of $\mathcal{P}$, each time after curtailing identified actively corrupt parties. The definition therefore is formalized with respect to a set $\mathcal{Q} \subseteq \mathcal{P}$.

Definition 6.3 ((α, β)-levelled sharing) *A value v is said to be (α, β)-levelled-shared with $\alpha \geq \beta$ amongst a set of parties $\mathcal{Q} \subseteq \mathcal{P}$ if every honest or passively corrupt party P_i in $\mathcal{Q}$ holds L_i as produced by $f_{\mathsf{LSh}}^{\alpha,\beta}(v)$ given in Figure 6.1.*

Function $f_{\mathsf{LSh}}^{\alpha,\beta}$

1. Choose uniformly random summands $s_\alpha, s_{\alpha-1}, \ldots s_\beta$ with $\sum_{i=\beta}^{\alpha} s_j = v$
2. For $j \in [\alpha, \beta]$, do the following:
   - Choose a random polynomial $g_j(x)$ of degree j with $g_j(0) = s_j$.
   - Sample the public parameter for eNICOM (Section 2.4.2.1) as $(\mathsf{epp}, t) \leftarrow \mathsf{eGen}(1^\kappa)$. For each share $s_{jk} = g_j(k)$, run $(c_{jk}, o_{jk}) \leftarrow \mathsf{eCom}(\mathsf{epp}, s_{jk}; r_{jk})$ $(P_k \in \mathcal{Q})$ where r_{jk} denotes randomness.
3. Set $L_i = \left(\{s_{ji}, o_{ji}\}_{j\in[\alpha,\beta]}, \{c_{jk}\}_{j\in[\alpha,\beta],P_k\in\mathcal{Q}}\right)$ for $P_i \in \mathcal{Q}$.

Figure 6.1: Function $f_{\mathsf{LSh}}^{\alpha,\beta}$ for computing (α, β)-levelled sharing

In our protocols the function $f_{\mathsf{LSh}}^{\alpha,\beta}$ will be realized via an MPC protocol, whereas, given the (α, β)-levelled-sharing, we will use a levelled-reconstruction protocol $\mathsf{LRec}^{\alpha,\beta}()$ that enforce reconstruction of the summands one at a time starting with s_α. This levelled reconstruction ensures a remarkable property tolerating any dynamic-admissible adversary– if the adversary can disrupt reconstruction of s_i, then it cannot learn s_{i-1} using its eavesdropping power. This property is instrumental in achieving fairness against the strong dynamic-admissible adversary. The protocol is presented in Figure 6.2. Its properties and round complexity are stated below. Note that starting with the feasibility condition $t_a + t_p < n = |\mathcal{P}|$, expelling a set of actively corrupt parties, say $\mathcal{B}$, makes the following impact on t_a, t_p and $\mathcal{P}$: $t_a = t_a - |\mathcal{B}|$, $t_p = t_p - |\mathcal{B}|$ and $\mathcal{P} = \mathcal{P} \setminus \mathcal{B}$. Consequently, the updated t_a, t_p and $\mathcal{P}$ continue to satisfy $t_a + t_p < |\mathcal{P}|$. Below, we will therefore use the fact that $t_a + t_p < |\mathcal{Q}|$, where $\mathcal{Q}$ denotes the relevant set of parties (i.e the set of parties remaining after possibly expelling a set of identified actively corrupt parties).

Protocol $\mathsf{LRec}^{\alpha,\beta}$

Inputs: Each P_i ($P_i \in \mathcal{Q}$) has input $L_i = \big(\{s_{ji}, o_{ji}\}_{j\in[\alpha,\beta]}, \{c_{jk}\}_{j\in[\alpha,\beta],P_k\in\mathcal{Q}}\big)$.

Output: Secret v or $\perp$ with set $\mathcal{B}$ constituting indices of the identified actively corrupt parties.

- For $j = \alpha$ down to β, P_i does the following round-by-round:
  - Broadcasts (s_{ji}, o_{ji}) and receive (s_{jk}, o_{jk}) from all $P_k \in \mathcal{Q}$ where $k \neq i$.
  - Initialize $\mathsf{Z}_j = i$ and populate Z_j in order to compute s_j as follows:
    - For each $k \neq i$, if commitment c_{jk} opens to s_{jk} via opening o_{jk}, then add k to Z_j.
    - If $|\mathsf{Z}_j| \geq j+1$, interpolate a j-degree polynomial $g_j(x)$ satisfying $g_j(k) = s_{jk}$ for $k \in \mathsf{Z}_j$ and compute $s_j = g_j(0)$. Else output $\perp$, set $\mathcal{B} = \mathcal{Q} \setminus \mathsf{Z}_j$ and terminate.
- Output $v = s_\alpha + \ldots s_\beta$.

Figure 6.2: Protocol $\mathsf{LRec}^{\alpha,\beta}$

Lemma 6.4 $\mathsf{LRec}^{\alpha,\beta}$ *satisfies the following properties–*

i. Correctness. *Each honest P_i participating in $\mathsf{LRec}^{\alpha,\beta}$ with input L_i as generated by $f^{\alpha,\beta}_{\mathsf{LSh}}(v)$, outputs either v or $\perp$ except with negligible probability.*

ii. Fault-Identification. *If an adversary disrupts the reconstruction of s_j, then $|\mathcal{B}| \geq |\mathcal{Q}| - j$.*

iii. Fairness. *If an adversary disrupts the reconstruction of s_j, then it does not learn s_{j-1}.*

iv. Round Complexity. *It terminates within $\alpha - \beta + 1$ rounds.*

Proof:

i. Consider an honest P_i participating with input $L_i = \big(\{s_{ji}, o_{ji}\}_{j\in[\alpha,\beta]}, \{c_{jk}\}_{j\in[\alpha,\beta],P_k\in\mathcal{Q}}\big)$. We observe P_i outputs $v' \neq \{v, \perp\}$ only if at least one of the summands, say $s_j (j \in [\alpha, \beta])$ is incorrectly set. This can happen only if P_i adds at least one index k to Z_j such that P_k sends an incorrect share $s'_{jk} \neq s_{jk}$. This occurs when (s'_{jk}, o'_{jk}) received from P_k is such that c_{jk} opens to s'_{jk} via o'_{jk} but $s'_{jk} \neq s_{jk}$. It now follows directly from the binding of eNICOM that this violation occurs with negligible probability. This completes the proof.

ii. Firstly, it follows from the property of Shamir-secret sharing and binding property of eNICOM that reconstruction of s_j would fail only if $|\mathsf{Z}_j| \leq j$. Next, note that as per the steps in Figure 6.2, each honest P_i would output $\mathcal{B} = \mathcal{Q} \setminus \mathsf{Z}_j$ if reconstruction of s_j fails. We can thus conclude that $|\mathcal{B}| = |\mathcal{Q}| - |\mathsf{Z}_j| \geq |\mathcal{Q}| - j$.

iii. To prove fairness, we first prove that if an adversary can disrupt the reconstruction of s_j, then it cannot learn s_{j-1} using its eavesdropping power. Since as per the protocol, the honest parties do not participate in the reconstruction of s_{j-1} when they fail to reconstruct s_j, the security of s_{j-1} follows from the information-theoretic security of Shamir-sharing and the statistical security (hiding) of eNICOM.

An adversary can disrupt reconstruction of s_j only if $|\mathsf{Z}_j| \leq j$. It is easy to check that Z_j would constitute the non-actively corrupt parties (honest and purely passive parties) i.e $\mathcal{Q}\setminus \subseteq \mathsf{Z}_j$. Thus, $|\mathcal{Q}\setminus| = |\mathcal{Q}| - t_a \leq |\mathsf{Z}_j| \leq j$. Lastly, to maintain $t_a + t_p < |\mathcal{Q}|$, it must hold that $t_p \leq |\mathcal{Q}| - t_a - 1 \leq j - 1$. Thus, the adversary corrupting $t_p \leq j-1$ parties cannot learn s_{j-1} using its eavesdropping power.

iv. The proof of the round complexity is straightforward. $\mathsf{LRec}^{\alpha,\beta}$ involves reconstruction of summands s_α down to s_β, each of which consumes one round; totalling upto $\alpha - \beta + 1$.

□

6.3.2 Upper bound for Fair MPC

The key insight for this protocol comes from [122] that builds on an MPC protocol with abort security to compute the function output in $(n-1,1)$-levelled-sharing form, followed by levelled-reconstruction to tackle dynamic corruption. Fairness is brought to the system by relying on the fairness of the levelled-reconstruction. In particular, the adversary is disabled to reconstruct $(i-1)$th summand, as a punitive action, when it disrupts reconstruction of the ith summand for the honest parties. In the marginal case, if the adversary disrupts the MPC protocol for computing the levelled-sharing and does not let the honest parties get their output, we disable it to reconstruct the $(n-1)$th summand itself.

In a (α,β)-levelled-reconstruction, the parameters α and β dictate the round complexity. The closer they are the better round complexity we obtain. The α and β in [122] are $n-2$ apart, shooting the round complexity of reconstruction to $n-1$. We depart from the construction of [122] in two ways to build a $(\lceil \frac{n}{2} \rceil + 1)$-round fair protocol. Firstly and prominently, we bring α and β much closer, cutting down $\lfloor \frac{n}{2} \rfloor$ summands from the levelled-secret sharing and bringing down the number of levels to just $n - 1 - \lfloor \frac{n}{2} \rfloor$ from $n-1$ of [122]. Second, we plug in the round-optimal (2-round) MPC protocol of [93, 35] achieving unanimous abort against malicious majority in the CRS model for computing the levelled-sharing of the output, making overall a $(\lceil \frac{n}{2} \rceil + 1)$-round fair protocol. We discuss the first departure in detail below.

Our innovation lies in fixing the best values of α and β without flouting fairness. The value of α and β, in essence determines the indispensable summands that we cannot do without.

Every possible *non-zero* threshold for active corruption maps to a crucial summand that the adversary using its corresponding admissible passive threshold cannot learn by itself, whilst the pool of non-disruptive set of parties, i.e. the set of honest and passive parties, can. This unique summand, being the 'soft spot' for the adversary, forces him to cooperate until the reconstruction of the immediate previous summand. As soon as the adversary does so, the honest parties turn self-reliant to compute the output, upholding fairness. We care only about the non-zero possibilities for the threshold of active corruption, as an all-passive adversary holds no power at its disposal to disrupt, leading to robust output reconstruction by all. For the minimum non-zero value of 1 active corruption, the unique summand is s_{n-2} that the adversary cannot learn using its admissible eavesdropping capacity of $n-2$, yet the set of non-disruptive parties, which is of size $n-1$, can. On the other extreme, for the maximum value of $\lceil \frac{n}{2} \rceil - 1$, the unique summand is $s_{\lfloor \frac{n}{2} \rfloor}$ that the adversary cannot learn using its admissible eavesdropping capacity of $\lfloor \frac{n}{2} \rfloor$, yet the set of non-disruptive parties, which is of size $\lfloor \frac{n}{2} \rfloor + 1$, can. This sets the values of α and β as $n-2$ and $\lfloor \frac{n}{2} \rfloor$ respectively, making the number of crucial summands only $\lceil \frac{n}{2} \rceil - 1$. The distance between these two parameters also captures the number of possible corruption scenarios with non-zero active corruption.

In the table below, we display for each admissible adversarial corruption (this set subsumes the crucial summands that we retain), whether the adversary and the set of non-disruptive parties respectively by themselves, can learn the summand, using its maximum eavesdropping capability and putting together their shares respectively. The pattern clearly displays the following feature: irrespective of the corruption scenario that the adversary follows, its maximum power to disrupt and eavesdrop remains one summand apart i.e. if it can disrupt ith summand with its maximum disruptive capability (and fall short of its power for failing the $(i-1)$th one), then its maximum eavesdropping capability does not allow it to learn $(i-1)$th summand by itself. Our fair protocol $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$ tolerating dynamic corruption appears in Figure 6.3. Assumption wise, $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$ relies on 2-round maliciously-secure OT in the common random/reference string model (when π_{UAbort} is instantiated with protocols of [93, 35]) and eNICOM (used in $\mathsf{LRec}^{\alpha,\beta}()$ and instantiated using Pedersen commitment scheme).

Table 6.1: Levelled-reconstruction where ($a = \mathtt{Y}/\mathtt{N}, b = \mathtt{Y}/\mathtt{N}$) under s_i indicates if $\mathcal{A}$ and non-active parties respectively can reconstruct s_i or not (Y = Yes, $\mathtt{N}$ = No)

$(t_a = \lvert\mathcal{D}\rvert, t_p = \lvert\mathcal{E}\rvert)$	$\lvert\mathcal{P} \setminus \mathcal{D}\rvert$	s_{n-2}	s_{n-3}	s_{n-4}		s_{n-i-1}		$s_{\lfloor n/2 \rfloor+1}$	$s_{\lfloor n/2 \rfloor}$
$(0, n-1)$	n	$(\mathtt{Y},\mathtt{Y})$	$(\mathtt{Y},\mathtt{Y})$	$(\mathtt{Y},\mathtt{Y})$	...	...	...	$(\mathtt{Y},\mathtt{Y})$	$(\mathtt{Y},\mathtt{Y})$
$(1, n-2)$	$n-1$	$(\mathtt{N},\mathtt{Y})$	$(\mathtt{Y},\mathtt{Y})$	$(\mathtt{Y},\mathtt{Y})$	...	...	...	$(\mathtt{Y},\mathtt{Y})$	$(\mathtt{Y},\mathtt{Y})$
$(2, n-3)$	$n-2$	$(\mathtt{N},\mathtt{N})$	$(\mathtt{N},\mathtt{Y})$	$(\mathtt{Y},\mathtt{Y})$	...	...	...	$(\mathtt{Y},\mathtt{Y})$	$(\mathtt{Y},\mathtt{Y})$
...	...	...	...	...	...	...	...	...	...
$(i, n-i-1)$	$n-i$	$(\mathtt{N},\mathtt{N})$	$(\mathtt{N},\mathtt{N})$	$(\mathtt{N},\mathtt{N})$	...	$(\mathtt{N},\mathtt{Y})$	...	$(\mathtt{Y},\mathtt{Y})$	$(\mathtt{Y},\mathtt{Y})$
...	...	...	...	...	...	...	...	...	...
$(\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$	$\lfloor n/2 \rfloor + 1$	$(\mathtt{N},\mathtt{N})$	$(\mathtt{N},\mathtt{N})$	$(\mathtt{N},\mathtt{N})$	...	...	...	$(\mathtt{N},\mathtt{N})$	$(\mathtt{N},\mathtt{Y})$

Protocol $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$

Inputs: Party P_j has x_j for $j \in [n]$

Building blocks: (a) Protocol π_{UAbort} achieving security with unanimous abort (realizing functionality $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2)) against malicious majority; (b) Protocol $\mathsf{LRec}^{\alpha,\beta}$ for reconstructing a (α, β)-levelled-shared value (Figure 6.2); (c) Function $f_{\mathsf{LSh}}^{n-2,\lfloor \frac{n}{2} \rfloor}$ (Figure 6.1).

Output: $y = f(x_1 \dots x_n)$ or $\perp$

Round $1-2$: Every P_j runs protocol π_{UAbort} to compute the function $f_{\mathsf{LSh}}^{n-2,\lfloor \frac{n}{2} \rfloor} \diamond f$ with input x_j to obtain L_j as the output. If $L_j = \perp$, it outputs $\perp$ and halts.

Round $3-(\lceil n/2 \rceil + 1)$: Each P_j participates in $\mathsf{LRec}^{n-2,\lfloor \frac{n}{2} \rfloor}$ with input L_j and outputs the outcome of $\mathsf{LRec}^{n-2,\lfloor \frac{n}{2} \rfloor}$.

Figure 6.3: Fair MPC against dynamic-admissible adversary

We state the formal theorem below.

Theorem 6.2 *Assuming the presence of a 2-round MPC protocol π_{UAbort} realizing $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2) against malicious-majority, protocol $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$ with n parties satisfies –*

- *Correctness: computes the correct output.*
- *Security: realizes $\mathcal{F}_{\mathsf{fair}}$ (Figure 2.3) against a dynamic-admissible $\mathcal{A}$ with threshold (t_a, t_p).*
- *Round complexity: runs in $\lceil n/2 \rceil + 1$ rounds.*

Proof: Correctness of $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$ follows directly from correctness of π_{UAbort} and $\mathsf{LRec}^{n-2,\lfloor \frac{n}{2} \rfloor}$ (Lemma 6.4). The security proof appears in Section 6.6.1.2. Round complexity of $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$ includes 2 rounds of π_{UAbort} and the round complexity of $\mathsf{LRec}^{n-2,\lfloor \frac{n}{2} \rfloor}$ which is $\left(n - 2 - \lfloor \frac{n}{2} \rfloor + 1\right) = \lceil n/2 \rceil - 1$ (Lemma 6.4); totalling upto $\lceil n/2 \rceil + 1$ rounds. □

6.3.3 Upper Bound for GOD MPC

At a broad level, robustness is achieved by rerunning our fair protocol as soon as failure occurs which can surface either in the underlying MPC or during reconstruction of any of the summands of the output. Taking inspiration from the player-elimination framework [118, 116], we maintain a history of deviating/disruptive behaviour across the runs and bar the identified parties from further participating. Such a paradigm calls for sequential runs and brings great challenge when round complexity is the concern. We hit the optimal round complexity banking on several ideas and interesting observations. First, we turn the underlying MPC protocol for computing (α, β)-levelled-sharing of the output to achieve *identifiability* so that any disruptive behaviour can be brought to notice. Slapping NIZK on the 2-round broadcast-only construction of [93] readily equips it with identifiability, without inflating the round complexity. Second, we leverage the *function-delayed* property of a modified variant of the protocol of [93] (proposed by [4]) where the first round messages are made independent of the function to be computed and the number of parties. This enables us to run many parallel instances (specifically $\lceil n/2 \rceil$) of the round 1 in the beginning and run the second round sequentially as and when failure happens to compute a new function each time as follows– (a) it hard-cores default input for the parties detected to be disruptive so far and (b) the output now is levelled-shared with new thresholds α and β each of which are smaller than the previous run by a function of the number of fresh catch, say δ. The latter brings the most crucial impact on the round complexity. Recall that the distance between α and β that impacts the round complexity, is directly coupled with the number of possible corruption scenarios with non-zero active corruption. Starting with the initial value of $\lceil \frac{n}{2} \rceil - 1$, each catch by δ reduces the number of possible corruption scenarios (with non-zero active corruption) and the distance between α and β by δ.

In the protocol, we maintain a number of dynamic variables which are updated during the run– (a) $\mathcal{L}$: the set of parties not identified to be actively corrupt and thus referred as alive; this set is initialized to $\mathcal{P}$; (b) $\mathcal{C}$: the set of parties identified as actively corrupt; this set initialized to $\emptyset$; (c) $\mathfrak{n}$: the parameter that dictates the number of corruption scenarios as $\lceil \frac{\mathfrak{n}}{2} \rceil$ and the possible corruption cases as $\{(0, \mathfrak{n}-1), \ldots, (\lceil \mathfrak{n}/2 \rceil - 1, \lfloor \mathfrak{n}/2 \rfloor)\}$; this is initialized to n that dictates the initial number of corruption cases as $\lceil \frac{n}{2} \rceil$ and the possible corruption cases as $\{(0, n-1), \ldots, (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)\}$. After every failure and a fresh catch of a set $\mathcal{B}$ of active corruptions, the sets $\mathcal{L}$, $\mathcal{C}$ and $\mathfrak{n}$ are updated as $\mathcal{L} = \mathcal{L} \setminus \mathcal{B}$, $\mathcal{C} = \mathcal{C} \cup \mathcal{B}$ and $\mathfrak{n} = \mathfrak{n} - 2|\mathcal{B}|$. The reduction of $\mathfrak{n}$ by $2|\mathcal{B}|$ denotes counting the reduction for active as well as passive corruptions. For every value of $\mathfrak{n}$, the formula for the total number of corruption scenarios, the values for (α, β) (that speaks about the indispensable summands as discussed in the fair protocol) and

the number of corruption scenarios with non-zero active corruption (which denotes the distance between (α, β)) remain the same– namely $\lceil \frac{\mathfrak{n}}{2} \rceil$, $(\mathfrak{n}-2, \lfloor \mathfrak{n}/2 \rfloor)$ and $\lceil \frac{\mathfrak{n}}{2} \rceil - 1$. In the marginal case, $\mathfrak{n}$ becomes either 1 or 2, the former when n is odd and all active corruptions are exposed making $(t_a, t_p) = (0, 0)$ and the latter when n is even and $(t_a, t_p) = (0, 1)$. With no active corruption in $\mathcal{L}$, the Round 2 of the MPC can be run to compute the output itself (instead of its levelled-sharing) robustly in both the marginal cases.

As the protocol follows an inductive behaviour based on $\mathfrak{n}$, to enable better understanding, we present below a snapshot of how the corruption scenarios shrinks after every catch of δ active corruption. The first column indicates a set of possible corruption scenarios, with (t_a, t_p) varying from $(0, \mathfrak{n}-1)$ to $(\lceil \mathfrak{n}/2 \rceil - 1, \lfloor \mathfrak{n}/2 \rfloor)$. If δ cheaters are identified, the first δ rows can simply be discarded as it is established that $t_a \geq \delta$. The number of feasible corruptions is thus slashed by δ. Next, these δ identified cheaters are eliminated, which reduces each (t_a, t_p) of the rows that sustained ($t_a = \delta$ onwards) by δ as shown by column 2. Finally, the column 3 displays column 2 with $\mathfrak{n}$ updated as $\mathfrak{n} - 2\delta$. The formal description of the protocol $\pi_{\mathsf{GOD}}^{\mathsf{dyn}}$ appears in Figure 6.4. Assumption wise, $\pi_{\mathsf{GOD}}^{\mathsf{dyn}}$ relies on 2-round maliciously-secure OT in the common random/reference string model, NIZK (when π_{idua} is instantiated with function-delayed variant of the protocol of [93] satisfying identifiability) and eNICOM (used in $\mathsf{LRec}^{\alpha,\beta}()$ and instantiated using Pedersen commitment scheme).

(t_a, t_p)	(t_a, t_p) after δ cheater identification	(t_a, t_p) after updating $\mathfrak{n} = \mathfrak{n} - 2\delta$
$(0, \mathfrak{n}-1)$	–	–
$(1, \mathfrak{n}-2)$	–	–
...	...	...
$(\delta, \mathfrak{n}-\delta-1)$	$(0, \mathfrak{n}-2\delta-1)$	$(0, \mathfrak{n}-1)$
$(\delta+1, \mathfrak{n}-\delta-2)$	$(1, \mathfrak{n}-2\delta-2)$	$(1, \mathfrak{n}-2)$
...	...	...
$(\lceil \mathfrak{n}/2 \rceil - 1, \lfloor \mathfrak{n}/2 \rfloor)$	$(\lceil \mathfrak{n}/2 \rceil - 1 - \delta, \lfloor \mathfrak{n}/2 \rfloor - \delta)$	$(\lceil \mathfrak{n}/2 \rceil - 1, \lfloor \mathfrak{n}/2 \rfloor)$

Protocol $\pi_{\mathsf{GOD}}^{\mathsf{dyn}}$

Inputs: Party P_i has x_i for $i \in [n]$

Building blocks: (a) Protocol π_{idua} achieving unanimous abort with identifiability (i.e. realizing functionality $\mathcal{F}_{\mathsf{idua}}$, refer Figure 2.5) against malicious majority and having function-delayed property; (b) Protocol $\mathsf{LRec}^{\alpha,\beta}$ for reconstructing a (α, β)-levelled-shared value (Figure 6.2); (c) Function $f_{\mathsf{LSh}}^{\alpha,\beta}$ (Figure 6.1).

Output: $y = f(x_1 \dots x_n)$

Step 1: P_i runs $\lceil n/2 \rceil$ parallel instances of Round 1 of π_{idua}, each using input x_i and independent randomness. Note that this round is independent of the function to be computed and number of parties. Initialize $k = 1$.

Step 2: Initialize, $\mathcal{L} = \mathcal{P}$, $\mathcal{C} = \emptyset$, $\mathfrak{n} = n$. Let $f^{\mathcal{C}}$ denote the function that is same as f except that the inputs of parties in $\mathcal{C}$ are hardcoded with default inputs. P_i executes the following steps:

2.1 If $\mathfrak{n} = 1, 2$, then run Round 2 of π_{idua} (considering kth instance of Round 1) among parties in $\mathcal{L}$ using input x_i to compute $f^{\mathcal{C}}$ and output the output of π_{idua} and terminate. (This corresponds to the case of no active corruptions.)

2.2 Run Round 2 of π_{idua} (considering kth instance of Round 1) among parties in $\mathcal{L}$ using input x_i to compute $f_{\mathsf{LSh}}^{\mathfrak{n}-2,\lfloor \frac{\mathfrak{n}}{2} \rfloor} \diamond f^{\mathcal{C}}$ and obtain L_i. If $L_i = \perp$ and $\mathcal{B}$ is set of parties identified to be corrupt, update $\mathcal{C} = \mathcal{C} \cup \mathcal{B}$, $\mathcal{L} = \mathcal{L} \setminus \mathcal{B}$, $\mathfrak{n} = \mathfrak{n} - 2|\mathcal{B}|$, $k = k + 1$ and repeat this step using updated value of $\mathfrak{n}$. Otherwise, participate in $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor \frac{\mathfrak{n}}{2} \rfloor}$ with input L_i. If $(\perp, \mathcal{B})$ is the output, then update $\mathcal{L}, \mathcal{C}, \mathfrak{n}, k$ as above and repeat this step using updated value of $\mathfrak{n}$. Otherwise, output the output of $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor \frac{\mathfrak{n}}{2} \rfloor}$ and terminate.

Figure 6.4: Robust MPC against dynamic-admissible adversary

We now analyze the round-complexity and security of $\pi_{\mathsf{GOD}}^{\mathsf{dyn}}$ below.

Lemma 6.5 *$\pi_{\mathsf{GOD}}^{\mathsf{dyn}}$ terminates in $\lceil n/2 \rceil + 1$ rounds.*

Proof: Consider an execution of $\pi_{\mathsf{GOD}}^{\mathsf{dyn}}$ (initialized with $\mathfrak{n} = n$). The outline of the proof is as follows: We give an inductive argument to prove the following - 'If Step 2 is executed with parameter $\mathfrak{n}$, then Step 2 terminates within $\lceil \frac{\mathfrak{n}}{2} \rceil$ rounds'. Assuming this claim holds, it follows directly that during the execution with $\mathfrak{n} = n$, Step 2 would terminate within $\lceil \frac{n}{2} \rceil$ rounds; thereby implying that the round complexity of $\pi_{\mathsf{GOD}}^{\mathsf{dyn}}$ is atmost $\lceil \frac{n}{2} \rceil + 1$ (adding the round for Step 1). We now prove the above claim by strong induction on $\mathfrak{n} \geq 1$.

Base Case ($\mathfrak{n} = 1, 2$): It follows directly from description in Figure 6.4 that Step 2 terminates in $\lceil \mathfrak{n}/2 \rceil = 1$ round when $\mathfrak{n} = 1, 2$.

Induction Hypothesis ($\mathfrak{n} \leq \ell$): Assume Step 2 terminates in $\lceil \mathfrak{n}/2 \rceil$ rounds for $\mathfrak{n} \leq \ell$.

Induction step ($\mathfrak{n} = \ell + 1$): Consider an execution of Step 2 with parameter $\mathfrak{n} = \ell + 1$. We analyze the following 3 exhaustive scenarios - (1) Suppose neither π_{idua} nor $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor \frac{\mathfrak{n}}{2} \rfloor}$ fails. (2) Suppose π_{idua} aborts. (3) Suppose π_{idua} does not abort but $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor \frac{\mathfrak{n}}{2} \rfloor}$ fails. We show that in each of them, Step 2 terminates within $\lceil \mathfrak{n}/2 \rceil = \lceil \frac{\ell+1}{2} \rceil$ rounds; thereby completing the induction step.

- Suppose neither π_{idua} nor $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor\frac{\mathfrak{n}}{2}\rfloor}$ fails. Then Step 2 involves following number of rounds– 1 (for Round 2 of π_{idua}) + number of rounds in $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor\frac{\mathfrak{n}}{2}\rfloor}$ i.e $(\mathfrak{n}-2-\lfloor\frac{\mathfrak{n}}{2}\rfloor+1)=\lceil\frac{\mathfrak{n}}{2}\rceil=\lceil(\ell+1)/2\rceil$ in total.

- Suppose π_{idua} aborts. Then $\mathcal{B}$ must comprise of at least one active party, implying that $\delta\geq 1$, where $\delta=|\mathcal{B}|$ and subsequently $\mathfrak{n}$ is updated to $\mathfrak{n}=(\mathfrak{n}-2\delta)\leq(\ell+1-2)=(\ell-1)$. Note that Step 2 now involves following number of rounds– 1 (for Round 2 of π_{idua}) + number of rounds in which Step 2 terminates when re-run with updated parameter $\mathfrak{n}$ i.e $\lceil\mathfrak{n}/2\rceil$ by induction hypothesis. Thus, the total number of rounds in Step 2 is $(1+\lceil\mathfrak{n}/2\rceil)\leq(1+\lceil\frac{\ell-1}{2}\rceil)=\lceil\frac{\ell+1}{2}\rceil$.

- Suppose π_{idua} does not abort but reconstruction $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor\frac{\mathfrak{n}}{2}\rfloor}$ fails. Say adversary disrupts reconstruction of summand $s_{\mathfrak{n}-r}$ in Round r of Step 2 (Round $r-1$ of $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor\mathfrak{n}/2\rfloor}$), where $r\in[2,\lceil\mathfrak{n}/2\rceil]$. It follows from fault identification property of Lemma 6.4 that $|\mathcal{B}|\geq|\mathcal{L}|-(\mathfrak{n}-r)\geq r$ (since $|\mathcal{L}|\geq\mathfrak{n}$ always holds). Consequently, $\delta=|\mathcal{B}|\geq r$ and updated parameter $\mathfrak{n}=\mathfrak{n}-2\delta\leq\ell+1-2r$. We now analyze the round complexity. Note that Step 2 involves following number of rounds– r (Reconstruction failed in Round $r\geq 2$ of Step 2 run with $\mathfrak{n}=\ell+1$) + number of rounds in which Step 2 terminates when re-run with updated parameter $\mathfrak{n}$ i.e $\lceil\mathfrak{n}/2\rceil$ by induction hypothesis. Thus total number of rounds in Step 2 is $(r+\lceil\mathfrak{n}/2\rceil)\leq(r+\lceil\frac{\ell+1-2r}{2}\rceil)=\lceil\frac{\ell+1}{2}\rceil$.

We point that induction hypothesis for $\mathfrak{n}=\mathfrak{n}-2\delta$ with $\delta\geq 1$ can be applied as $\mathfrak{n}\geq 1$ holds always in $\pi^{\mathsf{dyn}}_{\mathsf{GOD}}$ due to the following: the maximal value of δ is $\lceil\mathfrak{n}/2\rceil-1$ i.e the maximum possible number of actively corrupt parties. This completes the proof. □

Theorem 6.3 *Assuming the presence of a 2-round protocol π_{idua} realizing functionality $\mathcal{F}_{\mathsf{idua}}$ (Figure 2.5) against malicious majority and having function-delayed property; protocol $\pi^{\mathsf{dyn}}_{\mathsf{GOD}}$ with n parties satisfies–*

- *Correctness: computes the correct output.*
- *Security: realizes $\mathcal{F}_{\mathsf{god}}$ (Figure 2.4) against a dynamic-admissible $\mathcal{A}$ with threshold (t_a,t_p).*
- *Round complexity: runs in $\lceil n/2\rceil+1$ rounds.*

Proof: Correctness of $\pi^{\mathsf{dyn}}_{\mathsf{GOD}}$ follows directly from correctness of π_{idua} and correctness of $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor\frac{\mathfrak{n}}{2}\rfloor}$ (Lemma 6.4). The formal security proof appears in Section 6.6.2. Round complexity follows from Lemma 6.5. □

6.4 Lower Bounds for Boundary Corruption

In this section, we present two lower bounds for MPC protocol tolerating boundary-admissible adversaries and in the presence of CRS and PKI setup. Recall that such an adversary is restricted to corruption scenarios either $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ or $(t_a, t_p) = (0, n-1)$. We show that *three* and *four* rounds are necessary to achieve fairness and GOD respectively against a boundary-admissible adversary. It is to be noted that GOD is the de facto notion achieved in the pure passive corruption setting of $(t_a, t_p) = (0, n-1)$.

6.4.1 Impossibility of 3-round Robust MPC

In this section, we show that it is impossible to design a 3-round robust MPC protocol against boundary-admissible adversary with threshold (t_a, t_p) assuming both CRS and PKI. Notably, this lower bound is indeed surprising as the individual security guarantees translate to GOD against malicious-minority [108] and passive-majority [93, 35] for odd n (as $t_a = t_p$ wrt $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$), both of which are known to be attainable in just 2 rounds in the presence of CRS and PKI. Furthermore, it turns out interestingly that this lower bound does not hold against a boundary-admissble adversary with $t_a \leq 1$ (i.e boundary adversary corrupting with either $(t_a, t_p) = (1, \lfloor n/2 \rfloor)$ or $(t_a, t_p) = (0, n-1)$), and can be circumvented for this special case. In fact, we demonstrate a 3-round robust protocol in Section 6.5.3, against this special-case boundary-admissible adversary.

Theorem 6.4 *Assume parties have access to pairwise-private and broadcast channels, and a setup that includes CRS and PKI. Then, there exist functions f for which there is no 3-round protocol computing f that achieves guaranteed output delivery against boundary-admissible adversary.*

Proof: We prove the theorem for $n = 5$ parties. Let $\mathcal{P} = \{P_1, \ldots P_5\}$ denote the set of parties, where the adversary $\mathcal{A}$ may corrupt either with parametes $(t_a, t_p) = (2, 2)$ or $(t_a, t_p) = (0, 4)$. Here, the corruption scenarios translate to upto 2 active corruptions or upto 4 pure passive corruptions. We prove the theorem by contradiction. Suppose there exists a 3-round protocol π computing a common output function f that achieves GOD against such a boundary-admissible adversary.

At a high level, we discuss three adversarial strategies $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}_3$, where $\mathcal{A}_i$ is launched in an execution Σ_i of protocol π. While $\mathcal{A}_1, \mathcal{A}_2$ involve the case of active corruption of $\{P_1\}$ and $\{P_1, P_2\}$ respectively, $\mathcal{A}_3$ deals with the strategy of pure passive corruption of $\{P_1, P_3, P_4, P_5\}$. The executions are assumed to be run for the same input tuple $(x_1, x_2, x_3, x_4, x_5)$ and the same

random inputs $(r_1, r_2, r_3, r_4, r_5)$ of the parties. Let $\widetilde{x_i}$ denote the default input of P_i. (Same random inputs are considered for simplicity and without loss of generality. The same arguments hold for distribution ensembles as well.) First, when $\mathcal{A}_1$ is launched in Σ_1 we conclude that the output $\widetilde{y}$ at the end of the execution should be based on default input of P_1 and actual inputs of the remaining parties i.e $\widetilde{y} = f(\widetilde{x_1}, x_2, x_3, x_4, x_5)$. Next, strategy Σ_2 involving actively corrupt $\{P_1, P_2\}$ is designed such that corrupt P_2 obtains the same view in Σ_2 as an honest P_2 in Σ_1 and therefore computes the output $\widetilde{y}$ at the end of Σ_2. (Here, view of P_i includes x_i, r_i, the messages received during π and the knowledge related to CRS and PKI setup.) Lastly, a carefully designed strategy $\mathcal{A}_3$ by semi-honest parties $\{P_1, P_3, P_4, P_5\}$ allows $\mathcal{A}$ to obtain $\widetilde{y} = f(\widetilde{x_1}, x_2, x_3, x_4, x_5)$, in addition to the correct output i.e $y = f(x_1, x_2, x_3, x_4, x_5)$ at the end of execution Σ_3. This is a contradiction as it violates the security of π and can explicitly breach the privacy of honest P_2. This completes the proof overview.

We assume that the communication done in Round 2 and Round 3 of π is via broadcast alone. This holds without loss of generality since the parties can engage in point-to-point communication by exchanging random pads in the first round and then use these random pads to unmask later broadcasts. We use the following notation: Let $\mathsf{p}^1_{i\to j}$ denote the pairwise communication from P_i to P_j in round 1 and b^r_i denotes the broadcast by P_i in round r, where $r \in [3], \{i, j\} \in [5]$. These values may be function of CRS and the PKI setup as per the protocol specifications. Let V^ℓ_i denotes the view of party P_i at the end of execution Σ_ℓ ($\ell \in [3]$) of π. Below we describe the strategies $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}_3$.

$\mathcal{A}_1$: $\mathcal{A}$ corrupts $\{P_1\}$ actively here. P_1 behaves honestly in Round 1 and simply remains silent in Round 2 and Round 3.

$\mathcal{A}_2$: $\mathcal{A}$ corrupts $\{P_1, P_2\}$ actively here. The active misbehavior of P_1 is same as in $\mathcal{A}_1$ i.e P_1 behaves honestly in Round 1 and stops communicating thereafter. On the other hand, P_2 participates honestly upto Round 2 and remains silent in Round 3.

$\mathcal{A}_3$: $\mathcal{A}$ corrupts $\{P_1, P_3, P_4, P_5\}$ passively here. The semi-honest parties behave as per protocol specification throughout the execution Σ_3 to obtain the correct output. The passive strategy of $\{P_1, P_3, P_4, P_5\}$ is to ignore the Round 3 message from honest P_2 and locally compute the output based on the scenario of execution Σ_2 i.e imagining that P_1 stopped after Round 1 and P_2 stopped after Round 2.

We present a table depicting the views of the parties in executions Σ_1 and Σ_2 in Table 6.2. Here $\overline{\mathsf{b}^3_i}$ for $i \in \{2, 3, 4, 5\}$ denotes the message broadcast by honest P_i (as per its next-message function) in Round 3 in case P_1 behaves honestly in Round 1 but is silent in Round 2. The

views of parties in Σ_3 which is as per honest execution (since it involves only purely passive corruptions) appears in Table 6.3.

Table 6.2: Views of P_1, P_2, P_3, P_4, P_5 in Σ_1 and Σ_2

	Σ_1					Σ_2				
	V_1^1	V_2^1	V_3^1	V_4^1	V_5^1	V_1^2	V_2^2	V_3^2	V_4^2	V_5^2
Input	(x_1, r_1)	(x_2, r_2)	(x_3, r_3)	(x_4, r_4)	(x_5, r_5)	(x_1, r_1)	(x_2, r_2)	(x_3, r_3)	(x_4, r_4)	(x_5, r_5)
R1	$\mathsf{p}^1_{2\to1}$, $\mathsf{p}^1_{3\to1}$, $\mathsf{p}^1_{4\to1}$, $\mathsf{p}^1_{5\to1}$, b^1_2, b^1_3, b^1_4, b^1_5	$\mathsf{p}^1_{1\to2}$, $\mathsf{p}^1_{3\to2}$, $\mathsf{p}^1_{4\to2}$, $\mathsf{p}^1_{5\to2}$, b^1_1, b^1_3, b^1_4, b^1_5	$\mathsf{p}^1_{1\to3}$, $\mathsf{p}^1_{2\to3}$, $\mathsf{p}^1_{4\to3}$, $\mathsf{p}^1_{5\to3}$, b^1_1, b^1_2, b^1_4, b^1_5	$\mathsf{p}^1_{1\to4}$, $\mathsf{p}^1_{2\to4}$, $\mathsf{p}^1_{3\to4}$, , $\mathsf{p}^1_{5\to4}$, b^1_1, b^1_2, b^1_3, b^1_5	$\mathsf{p}^1_{1\to5}$, $\mathsf{p}^1_{2\to5}$, $\mathsf{p}^1_{3\to5}$, , $\mathsf{p}^1_{4\to5}$, b^1_1, b^1_2, b^1_3, b^1_4	$\mathsf{p}^1_{2\to1}$, $\mathsf{p}^1_{3\to1}$, $\mathsf{p}^1_{4\to1}$, $\mathsf{p}^1_{5\to1}$, b^1_2, b^1_3, b^1_4, b^1_5	$\mathsf{p}^1_{1\to2}$, $\mathsf{p}^1_{3\to2}$, $\mathsf{p}^1_{4\to2}$, $\mathsf{p}^1_{5\to2}$, b^1_1, b^1_3, b^1_4, b^1_5	$\mathsf{p}^1_{1\to3}$, $\mathsf{p}^1_{2\to3}$, $\mathsf{p}^1_{4\to3}$, $\mathsf{p}^1_{5\to3}$, b^1_1, b^1_2, b^1_4, b^1_5	$\mathsf{p}^1_{1\to4}$, $\mathsf{p}^1_{2\to4}$, $\mathsf{p}^1_{3\to4}$, $\mathsf{p}^1_{5\to4}$, b^1_1, b^1_2, b^1_3, b^1_5	$\mathsf{p}^1_{1\to5}$, $\mathsf{p}^1_{2\to5}$, $\mathsf{p}^1_{3\to5}$, $\mathsf{p}^1_{4\to5}$, b^1_1, b^1_2, b^1_3, b^1_4
R2	b^2_2, b^2_3, b^2_4, b^2_5	–, b^2_3, b^2_4, b^2_5	–, b^2_2, b^2_4, b^2_5	–, b^2_2, b^2_3, b^2_5	–, b^2_2, b^2_3, b^2_4	b^2_2, b^2_3, b^2_4, b^2_5	–, b^2_3, b^2_4, b^2_5	–, b^2_2, b^2_4, b^2_5	–, b^2_2, b^2_3, b^2_5	–, b^2_2, b^2_3, b^2_4
R3	$\overline{\mathsf{b}^3_2}$, $\overline{\mathsf{b}^3_3}$, $\overline{\mathsf{b}^3_4}$, $\overline{\mathsf{b}^3_5}$	–, $\overline{\mathsf{b}^3_3}$, $\overline{\mathsf{b}^3_4}$, $\overline{\mathsf{b}^3_5}$	–, $\overline{\mathsf{b}^3_2}$, $\overline{\mathsf{b}^3_4}$, $\overline{\mathsf{b}^3_5}$	–, $\overline{\mathsf{b}^3_2}$, $\overline{\mathsf{b}^3_3}$, $\overline{\mathsf{b}^3_5}$	–, $\overline{\mathsf{b}^3_2}$, $\overline{\mathsf{b}^3_3}$, $\overline{\mathsf{b}^3_4}$	–, $\overline{\mathsf{b}^3_3}$, $\overline{\mathsf{b}^3_4}$, $\overline{\mathsf{b}^3_5}$	–, $\overline{\mathsf{b}^3_3}$, $\overline{\mathsf{b}^3_4}$, $\overline{\mathsf{b}^3_5}$	–, –, $\overline{\mathsf{b}^3_4}$, $\overline{\mathsf{b}^3_5}$	–, –, $\overline{\mathsf{b}^3_3}$, $\overline{\mathsf{b}^3_5}$	–, –, $\overline{\mathsf{b}^3_3}$, $\overline{\mathsf{b}^3_4}$

Table 6.3: Views of P_1, P_2, P_3, P_4, P_5 in Σ_3

	Σ_3				
	V_1^1	V_2^1	V_3^1	V_4^1	V_5^1
Input	(x_1, r_1)	(x_2, r_2)	(x_3, r_3)	(x_4, r_4)	(x_5, r_5)
R1	$\mathsf{p}^1_{2\to1}$, $\mathsf{p}^1_{3\to1}$, $\mathsf{p}^1_{4\to1}$, $\mathsf{p}^1_{5\to1}$, b^1_2, b^1_3, b^1_4, b^1_5	$\mathsf{p}^1_{1\to2}$, $\mathsf{p}^1_{3\to2}$, $\mathsf{p}^1_{4\to2}$, $\mathsf{p}^1_{5\to2}$, b^1_1, b^1_3, b^1_4, b^1_5	$\mathsf{p}^1_{1\to3}$, $\mathsf{p}^1_{2\to3}$, $\mathsf{p}^1_{4\to3}$, $\mathsf{p}^1_{5\to3}$, b^1_1, b^1_2, b^1_4, b^1_5	$\mathsf{p}^1_{1\to4}$, $\mathsf{p}^1_{2\to4}$, $\mathsf{p}^1_{3\to4}$, $\mathsf{p}^1_{5\to4}$, b^1_1, b^1_2, b^1_3, b^1_5	$\mathsf{p}^1_{1\to5}$, $\mathsf{p}^1_{2\to5}$, $\mathsf{p}^1_{3\to5}$, $\mathsf{p}^1_{4\to5}$, b^1_1, b^1_2, b^1_3, b^1_4
R2	b^2_2, b^2_3, b^2_4, b^2_5	b^2_1, b^2_3, b^2_4, b^2_5	b^2_1, b^2_2, b^2_4, b^2_5	b^2_1, b^2_2, b^2_3, b^2_5	b^2_1, b^2_2, b^2_3, b^2_4
R3	b^3_2, b^3_3, b^3_4, b^3_5	b^3_1, b^3_3, b^3_4, b^3_5	b^2_1, b^3_2, b^3_4, b^3_5	b^3_1, b^3_2, b^3_3, b^3_5	b^3_1, b^3_2, b^3_3, b^3_4

We now present a sequence of lemmas to complete the proof.

Lemma 6.6 *At the end of Σ_1, parties compute output $\widetilde{y} = f(\widetilde{x_1}, x_2, x_3, x_4, x_5)$, where $\widetilde{x_1}$ denotes the default input of P_1.*

Proof: Firstly, since Σ_1 involves active behavior only by P_1, it follows directly from correctness and robustness of π that the output computed at the end of Σ_1, say y' should be based on actual

inputs x_i for $i \in \{2,3,4,5\}$. Now, there are two possibilities with respect to input of P_1 i.e y' is based on either x_1 (i.e the input used by P_1 in Round 1 of Σ_1) or $\widetilde{x_1}$ (default input). In case of the latter, the lemma holds directly. We now assume the former for contradiction.

Suppose the output y' is based on x_1 rather than $\widetilde{x_1}$. Since P_1 stops communicating after Round 1, we can conclude that the combined views of $\{P_2, P_3, P_4, P_5\}$ must suffice to compute the output $y' = f(x_1, \ldots, x_5)$ at the end of Round 1 itself. If this holds, we argue that π cannot be secure as follows: Suppose π is such that when all parties participate honestly in Round 1, the combined view of $\{P_2, P_3, P_4, P_5\}$ suffices to compute the output at the end of Round 1 itself. Then, in an execution of π, an adversary corrupting $\{P_2, P_3, P_4, P_5\}$ purely passively (correponding to $(t_a, t_p) = (0, 4)$) can learn the output on various inputs of its choice, keeping x_1 fixed. This residual attack breaches privacy of honest P_1 (A concrete example of such an f appears at the end of this section). We have thus arrived at a contradiction. This completes the proof that y' must be based on $\widetilde{x_1}$, rather than x_1 and consequently $y' = \widetilde{y} = f(\widetilde{x_1}, x_2, x_3, x_4, x_5)$ must be the output computed at the end of Σ_1. □

Lemma 6.7 *At the end of Σ_2, parties compute output $\widetilde{y} = f(\widetilde{x_1}, x_2, x_3, x_4, x_5)$, where $\widetilde{x_1}$ denotes the default input of P_1.*

Proof: Recall that $\mathcal{A}_2$ is similar to $\mathcal{A}_1$ involving active P_1, except that P_2 is active as well with the strategy of behaving honestly upto Round 2 and remaining silent in Round 3. Since the executions Σ_1 and Σ_2 proceed identically upto Round 2, it is easy to check that the view of corrupt P_2 in Σ_2 is same as honest P_2 in Σ_1 (refer to Table 6.2). It now follows directly from Lemma 6.6 that P_2 computes the output $\widetilde{y} = f(\widetilde{x_1}, x_2, x_3, x_4, x_5)$. By correctness and robustness of π computing the common output function f, it must hold that all parties output $\widetilde{y}$ at the end of Σ_2. □

Lemma 6.8 *The combined view of parties $\{P_3, P_4, P_5\}$ at the end of Round 2 of Σ_2 suffices to compute the output of Σ_2 i.e $\widetilde{y}$.*

Proof: We note that as per $\mathcal{A}_2$, both $\{P_1, P_2\}$ do not communicate in Round 3; implying that the combined view of honest parties $\{P_3, P_4, P_5\}$ at the end of Round 2 of Σ_2 must suffice to compute the output of Σ_2 i.e $\widetilde{y}$ (Lemma 6.7). □

Lemma 6.9 *An adversary executing strategy $\mathcal{A}_3$ obtains the value $\widetilde{y} = f(\widetilde{x_1}, x_2, x_3, x_4, x_5)$, in addition to the correct output $y = f(x_1, x_2, x_3, x_4, x_5)$ at the end of Σ_3.*

Proof: Firstly, Σ_3 must lead to computation of correct output i.e $y = f(x_1, x_2, x_3, x_4, x_5)$ by all parties since $\mathcal{A}_3$ involves only semi-honest corruptions. Next, it is easy to check from Tables

6.2 and 6.3 that the combined view of adversary corrupting $\{P_1, P_3, P_4, P_5\}$ passively at the end of Round 2 of Σ_3 subsumes the combined view of honest parties $\{P_3, P_4, P_5\}$ at the end of Round 2 of Σ_2. It now follows directly from Lemma 6.8 that the adversary can obtain the output $\widetilde{y}$ as well.

In more detail, $\mathcal{A}$ launching $\mathcal{A}_3$ in Σ_3 can compute the output as per the scenario of Σ_2 as follows- Let $\overline{\mathsf{b}_i^3}$ for $i \in \{2,3,4,5\}$ denotes the message broadcast by honest P_i (as per its next-message function) in Round 3 in case P_1 behaves honestly in Round 1 but is silent in Round 2. Locally compute $\{\overline{\mathsf{b}_3^3}, \overline{\mathsf{b}_4^3}, \overline{\mathsf{b}_5^3}\}$ ($\overline{\mathsf{b}_i^3}$ is a function of P_i's ($i \in \{3,4,5\}$) view at the end of Round 2) by imagining that P_1 did not send Round 2 message and compute $\widetilde{y}$ by ignoring the message sent by honest P_2 in Round 3. Thus, by following strategy $\mathcal{A}_3$, $\mathcal{A}$ obtains multiple evaluations of f i.e both y and $\widetilde{y}$ which violates the security of π. (We give a concrete example of such an f below that breaches privacy of honest P_2.) This completes the proof of the lemma. □

Thus, we have arrived at a contradiction to our assumption that π is secure. While the above proof was shown specifically for $n = 5$, it can be extended to any $n > 5$ in the following natural manner: The strategies $\mathcal{A}_1, \mathcal{A}_2$ remain the same (feasible as atleast two active corruptions are allowed when $n > 5$) and let us conclude that the combined view of $\{P_3, P_4 \dots P_n\}$ at the end of Round 2 suffices to compute $\widetilde{y} = f(\widetilde{x_1}, x_2 \dots x_n)$. Accordingly, strategy $\mathcal{A}_3$ involving passive corruption of $\{P_1, P_3, P_4 \dots P_n\}$ would lead to the adversary obtaining multiple evaluations of the function leading to the final contradiction. This completes the proof of Theorem 6.4. □

Next, we give a concrete example of f to demonstrate how the strategy of residual attack used to prove the lower bound in Theorem 6.4.

Concrete Example of f: Let $f(x_1, x_2, x_3, x_4, x_5)$ with $x_1 = (\alpha, \beta), x_2 = (b, m_0, m_1)$ (where α, β, b are single bit values) and $x_3 = x_4 = x_5 = \perp$ be defined as below for P_i's input x_i:

$$f(x_1, x_2, x_3, x_4, x_5) = \begin{cases} m_\alpha & \text{if } b = 0 \\ m_{\alpha \oplus \beta} & \text{otherwise} \end{cases}$$

Using this function f, we describe explicitly how multiple evaluations of f breaches privacy of P_1 and P_2 in the argument of Lemma 6.6 and Lemma 6.8 respectively. Consider the adversary corrupting $\{P_2, P_3, P_4, P_5\}$ passively ($(t_a, t_p) = (0, 4)$) that can learn the output on various inputs of its choice, keeping x_1 fixed (in Lemma 6.6). By locally plugging in inputs $b = 0$ and $b = 1$ on behalf of passive P_2, it is easy to check that the adversary can learn both α and β. This violates privacy of honest P_1 as its input β is never revealed as per the ideal functionality. Next, consider the adversary of Lemma 6.8 corrupting $\{P_1, P_3, P_4, P_5\}$ who obtains both $y = f(x_1, x_2, x_3, x_4, x_5)$ and $\widetilde{y} = f(\widetilde{x_1}, x_2, x_3, x_4, x_5)$. We claim this breaches privacy of honest P_2

as follows: As per the ideal functionality, the adversary would learn exactly only one among m_0, m_1. Next, suppose the default value of $\widetilde{x_1} = (0,0)$. Then by participating in Σ_3 with input $x_1 = (1,0)$, the adversary would obtain both $y = m_1$ and $\widetilde{y} = m_0$ (irrespective of b) which compromises the security of honest P_2's input.

Before concluding this section, we give quick intuition of why the above lower bound argument does not hold when malicious corruption $t_a \leq 1$. Note that the strategy $\mathcal{A}_3$ carried out by the adversary corrupting $\{P_1, P_3, P_4, P_5\}$ purely passively was feasible only since the output on default input of P_1 could be computed without any dependency on honest P_2's message in Round 3. Had it been the case that honest P_2's Round 3 message was crucial for output computation, then the semi-honest parties $\{P_1, P_3, P_4, P_5\}$ would have obtained only the output on the combination of actual inputs and would be unable to breach security. Tracing back, recall that the partnership of malicious $\{P_1, P_2\}$ together in $\mathcal{A}_2$ was crucial in implying this non-dependency on Round 3 message of P_2 (which led us to the conclusion of view of $\{P_3, P_4, P_5\}$ being sufficient to compute output on P_1's default input). It is thereby evident that without such a partnership of two malicious parties, it would not be possible to arrive at such a contradiction. This intuition is further substantiated by our 3-round upper bound achieving GOD in case of single active corruption (Section 6.5.3).

6.4.2 Impossibility of 2-round Fair MPC

We begin with the observation that the existing 3-round lower bounds of [102, 108, 166] for fair malicious-minority MPC do not carry over in our setting. The lower bound of both [102, 108] break down when the parties have access to a PKI (as acknowledged/demonstrated in their work). The result of [166], assuming access to pairwise-private and broadcast channels, also breaks down when parties have access to a PKI. The proof, originally given without the mention of CRS, seems to withstand a CRS. The proof approach of [166] is via contradiction i.e derives a series of implications assuming that 2-round fair MPC protocol π exists and eventually builds up to a contradiction. A crucial lemma in their proof (Lemma 24 in their full version [168]) states that π must be such that a single party, say P_1, is able to compute the output at the end of Round 1. The argument for this claim relies on the fact that (a) the adversary's communication stops after Round 1 and (b) the Round 2 messages of honest parties do not hold any potential useful information to aid P_1's output computation. Roughly speaking, (b) follows since the honest party's messages are fully determined by the information available to P_1 at the end of Round 1 itself and can therefore be locally computed by P_1. This information includes the broadcast communication by the adversary in Round 1. While the above argument regarding (b) holds in the plain model and even in the presence of public setup such as CRS,

it does not hold in the presence of private setup like PKI. In this case, an honest party may hold some private information unknown to P_1 at the end of Round 1, such as the decryption of the adversary's Round 1 broadcast using its exclusive secret key; which may aid in output computation by P_1. Consequently, this claim of [166] and their proof are not resilient to the presence of PKI. We now present our lower bound formally.

Theorem 6.5 *There exist functions f for which there is no 2-round n-party MPC protocol that achieves fairness against boundary-admissible adversary, in a setting with pairwise-private and broadcast channels, and a setup that includes CRS and PKI.*

Proof: We prove the theorem for $n = 3$ parties, where boundary-admissible adversary $\mathcal{A}$ chooses corruption parameters either $(t_a, t_p) = (1, 1)$ or $(t_a, t_p) = (0, 2)$. Here, the corruption scenarios translate to either upto 1 active corruption or upto 2 purely passive corruptions. Let $\{P_1, P_2, P_3\}$ denote the set of parties with P_i having input x_i. Suppose by contradiction, π is an MPC protocol computing f that achieves fairness against $\mathcal{A}$. To be more specific, π is fair if $(t_a, t_p) = (1, 1)$ and achieves GOD otherwise (as GOD is the de-facto security guarantee incase of no active corruptions i.e $(t_a, t_p) = (0, 2)$). On a high-level, we first exploit fairness of π to conclude that the combined view of a set of 2 parties suffices for output computation at the end of Round 1. (Here, view of P_i includes x_i, its randomness r_i, the messages received during π and the knowledge related to CRS and PKI setup.) Next, considering a strategy where the adversary $\mathcal{A}$ corrupts this set of 2 parties purely passively leads us to conclude that $\mathcal{A}$ can compute the output at the end of Round 1 itself; leading upto a final contradiction. We now present the sequence of claims to complete the formal proof.

Lemma 6.10 *Protocol π must be such that the combined view of $\{P_2, P_3\}$ at the end of Round 1 suffices for output computation.*

Proof: The proof of the lemma is straightforward. Assume $\mathcal{A}$ corrupting P_1 actively (with $(t_a, t_p) = (1, 1)$) with the following strategy: P_1 behaves honestly in Round 1 and simply remains silent in Round 2. It is easy to check that P_1 would obtain the output due to correctness of π, as he receives the entire protocol communication as per honest execution. Since π is fair, the honest parties $\{P_2, P_3\}$ must also obtain the output at the end of π; even without P_1's communication in Round 2. Thus, we conclude that the combined view of $\{P_2, P_3\}$ at the end of Round 1 suffices for output computation. □

Lemma 6.11 *There exists an adversarial strategy such that the adversary obtains the output at the end of Round 1.*

Proof: The proof follows directly from Lemma 6.10– $\mathcal{A}$ corrupting $\{P_2, P_3\}$ purely passively ($(t_a, t_p) = (0, 2)$) would obtain the output at the end of Round 1. □

Lemma 6.12 *Protocol π does not achieve privacy.*

Proof: It is implied from Lemma 6.11 that $\mathcal{A}$ corrupting $\{P_2, P_3\}$ purely passively can obtain multiple evaluations of the function f by locally plugging in different values for $\{x_2, x_3\}$ while honest P_1's input x_1 remains fixed. This 'residual function attack' violates privacy of P_1. We refer to the argument in Lemma 6.3 for a concrete example. □

We have arrived at a contradiction, concluding the proof of Theorem 6.5. It is easy to check that this argument can be extended for higher values of n. □

6.5 Upper bounds for Boundary Corruption

In this section, we describe three upper bounds with respect to the boundary-admissible adversary $\mathcal{A}$ with threshold (t_a, t_p). We first present a robust upper bound in 4 rounds for the general case. Next, we present a 3-round robust protocol for the special case of single active corruption, which circumvents our lower bound of Section 6.4.1. Finally, we present our fair 3-round upper bound that can be arrived at by simplifying the robust general-case construction. Note that even the fair construction is robust in the corruption scenario of no active corruptions i.e $(t_a, t_p) = (0, n-1)$. The security guarantees differ only in case of corruption scenario involving malicious corruptions. All the above three constructions are round-optimal, following our lower bound results of Section 6.4.1 and 6.4.2. We start with a building block commonly used across all our constructs.

6.5.1 Authenticated Secret Sharing

We introduce the primitive of Authenticated Secret Sharing [130, 127] used in our upper bounds against the boundary-admissible $\mathcal{A}$.

Definition 6.4 (α-authenticated sharing) *A value v is said to be α-authenticated-shared amongst a set of parties $\mathcal{P}$ if every honest or passively corrupt party P_i in $\mathcal{P}$ holds S_i as produced by $f^{\alpha}_{\mathsf{ASh}}(v)$ given in Figure 6.5.*

Function $f^{\alpha}_{\mathsf{ASh}}(v)$

1. *α shamir-sharing of secret v:* Choose random $a_1, a_2 \ldots a_\alpha \in \mathbb{F}$, where $\mathbb{F}$ denotes a finite field. Build the α-degree polynomial $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{\alpha-1} x^{\alpha-1} + a_\alpha x^\alpha$, where $a_0 = v$. Let $\mathsf{sh}_i = A(i)$ for $i \in [n]$.

2. *Authentication of shares:* For all $i, j \in [n]$, choose random one-time message-authentication codes (MAC) [105] keys $k_{ij} \in \{0,1\}^\kappa$ and compute $\mathsf{tag}_{ij} = \mathsf{Mac}_{k_{ij}}(\mathsf{sh}_i)$.

3. Output $S_i = \left(\mathsf{sh}_i, \{k_{ji}\}_{j\in[n]}, \{\mathsf{tag}_{ij}\}_{j\in[n]}\right)$ for $i \in [n]$.

Figure 6.5: Function for Authenticated secret-sharing

In our upper bounds, the function $f^{\alpha}_{\mathsf{ASh}}$ is realized via MPC protocols. The reconstruction will be done via protocol ARec^{α} (Figure 6.6) amongst the parties. We prove the relevant properties below:

Protocol ARec^{α}

Inputs: P_i participates with $S_i = \left(\mathsf{sh}_i, \{k_{ji}\}_{j\in[n]}, \{\mathsf{tag}_{ij}\}_{j\in[n]}\right)$

Output: Secret v'

Each P_i does the following:

1. Broadcast $\left(\mathsf{sh}_i, \{\mathsf{tag}_{ij}\}_{j\in[n]}\right)$ and receive $\left(\mathsf{sh}'_j, \mathsf{tag}'_{ji}\right)$ from $j \neq i$.

2. Each P_i outputs v' as follows:

   - Initialize $\mathcal{V}$ to $\{i\}$. For $j \neq i$, if $\mathsf{Mac}_{k_{ji}}(\mathsf{sh}'_j) = \mathsf{tag}'_{ji}$, set $\mathsf{sh}_j = \mathsf{sh}'_j$ and add j to $\mathcal{V}$; else set $\mathsf{sh}_j = \perp$.
   - If $|\mathcal{V}| \geq \alpha + 1$, interpolate a α degree polynomial $A'(x)$ satisfying $A'(\gamma) = \mathsf{sh}_\gamma$ for $\gamma \in \mathcal{V}$. Output $\perp$ if the above fails, else output $v' = A'(0)$.

Figure 6.6: Protocol for Reconstruction of an authenticated-secret

Lemma 6.13 *The pair* $(f^{\alpha}_{\mathsf{ASh}}, \mathsf{ARec}^{\alpha})$ *satisfies the following:*

i. Privacy. *For all* $v \in \mathbb{F}$*, the output* $(S_1, \ldots, S_n) \leftarrow f^{\alpha}_{\mathsf{ASh}}(v)$ *satisfies the following– for all* $\{i_1, \ldots i_{\alpha'}\} \subset [n]$ *with* $\alpha' \leq \alpha$*, the distribution of* $\{S_{i_1}, \ldots, S_{i_{\alpha'}}\}$ *is statistically independent of* v*.*

ii. Correctness. *For all* $v \in \mathbb{F}$*, the value* v' *output by all honest parties at the end of* $\mathsf{ARec}^{\alpha}(S'_1, \ldots S'_n)$ *satisfies the following– For all* $(S_1, \ldots, S_n) \leftarrow f^{\alpha}_{\mathsf{ASh}}(v)$ *and* $(S'_1, \ldots, S'_n)$ *such that* $S'_i = S_i$ *corresponding to atleast* $\alpha + 1$ *parties* P_i*, it holds that* $\Pr[v' \neq v] \leq \mathtt{negl}(\kappa)$ *for a computational security parameter* κ*.*

iii. Round complexity. ARec^{α} *terminates in one round.*

Proof:

i. Privacy: It is easy to check from the description of $f^{\alpha}_{\mathsf{ASh}}$ that privacy follows directly from the fact that v is Shamir-shared with degree α.

ii. Correctness: Firstly, we note that since atleast $\alpha+1$ parties P_i participate with $S'_i = S_i$ in $\mathsf{ARec}^{\alpha}(S'_1, \ldots S'_n)$, the $\mathcal{V}$ set of each honest party comprises of atleast $(\alpha+1)$ correct shares sh_i. These shares suffice to uniquely reconstruct the α-shared secret v. We can thus conclude that an honest P_i would output $\perp$ only if the interpolation of the α-degree polynomial fails, which in turn occurs if there is an incorrect share, say sh'_j, such that j is added to $\mathcal{V}$. This would imply that a corrupt P_j broadcasts $\mathsf{sh}'_j \neq \mathsf{sh}_j$ and tag'_{ji} but satisfied the condition $\mathsf{Mac}_{k_{ji}}(\mathsf{sh}'_j) = \mathsf{tag}'_{ji}$, with respect to the MAC-key k_{ji} (present in S_i) available to honest P_i (not to P_j). However, security of MAC ensure that the above cannot happen except with negligible probability. This completes the proof of correctness.

iii. Round complexity. The proof is self-evident.

□

6.5.2 Upper bound for Robust MPC: The general case

In a setting where either at most $n-1$ passive corruption or at most $(\lceil \frac{n}{2} \rceil - 1)$ active corruption takes place, [127] presents a protocol relying on two types of MPC protocol. An actively-secure protocol against malicious majority is used to compute an authenticated-sharing of the output with threshold $(\lceil \frac{n}{2} \rceil - 1)$. When this protocol succeeds, the output is computed via reconstruction of the authenticated-sharing. On the other hand, a failure is tackled via running a robust honest-majority (majority of the parties are honest) actively-secure protocol, relying on the conclusion that the protocol is facing a malicious-minority. When n is odd, we need to tackle the exact corruption scenarios as that of the protocols of [127]. On the other hand when n is even, the extreme case for active corruption accommodates an additional passive corruption. Apart from hitting optimal round complexity, tackling the distinct boundary cases for odd and even n in a unified way brings challenge for our protocol.

We make the following effective changes to the approach of [127]. First, we invoke a 2-round actively-secure protocol π_{idua} with identifiable abort against malicious majority (can be instantiated with protocols of [93, 35] augmented with NIZKs) to compute $\lfloor \frac{n}{2} \rfloor$-authenticated-sharing of the output. When we expel the identified corrupt parties in case of failure (which may occur in corruption scenario $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$), the remaining population always displays honest-majority, no matter whether n is odd or even (For instance, elimination of 1 corrupt party results in $t' \leq (t_p - 1) = \lfloor n/2 \rfloor - 1$ total corruptions among $n' = (n-1)$ remaining parties which satisfies $n' \geq 2t' + 1$.). The robust honest-majority protocol π_{GOD} is then invoked

to compute the function f where the inputs of the identified parties are hard-coded to default values. The change in the degree of authenticated sharing ensures that an adversary choosing to corrupt in the boundary case of $\lceil \frac{n}{2} \rceil - 1$ active corruption and zero (when n is odd) or one (when n is even) purely passive corruption, cannot learn the output by itself collating the information it gathers during π_{idua}. Without the change, the adversary could ensure that π_{idua} leads to a failure for the honest parties and yet could learn outputs from both π_{idua} and π_{GOD} with different set of adversarial-inputs. Lastly, the function and input independence property of Round 1 of the 3-round honest-majority protocol of [108, 4] allows us to superimpose this round with the run of π_{idua}. Both these instantations of π_{GOD} are also equipped to tackle the probable change in population for the remaining two rounds (when identified corrupt parties are expelled) and the change in the function to be computed (with hard-coded default inputs for the identified corrupt parties). Our protocol appears in Figure 6.7. Assumption wise, $\pi^{\mathsf{bou}}_{\mathsf{GOD}}$ relies on 2-round maliciously-secure OT in the common random/reference string model, NIZK (when π_{idua} is instantiated with function-delayed variant of the protocol of [93] satisfying identifiability) and Zaps and public-key encryption (when π_{GOD} is instantiated with the protocol of [4]).

Protocol $\pi^{\mathsf{bou}}_{\mathsf{GOD}}$

Inputs: Party P_i has x_i for $i \in [n]$

Building Blocks: (a) 2-round protocol π_{idua} achieving identifiable abort against malicious majority (realizing functionality $\mathcal{F}_{\mathsf{idua}}$, refer Figure 2.5); (b) 3-round honest-majority actively-secure robust protocol π_{GOD} (realizing functionality $\mathcal{F}_{\mathsf{god}}$, refer Figure 2.4) with additional property of Round 1 being function and input independent; (c) Protocol $\mathsf{ARec}^{\lfloor n/2 \rfloor}$ for reconstructing an $\lfloor n/2 \rfloor$-authenticated-shared secret (Figure 6.6); (d) Function $f^{\lfloor n/2 \rfloor}_{\mathsf{ASh}}$ (Figure 6.5).

Output: $y = f(x_1 \dots x_n)$

Round 1–2: The parties run π_{idua} computing the function $f^{\lfloor n/2 \rfloor}_{\mathsf{ASh}} \diamond f$ with input x_i to obtain output $(S_i = (\mathsf{sh}_i, \{k_{ji}\}_{j\in[n]}, \{\mathsf{tag}_{ij}\}_{j\in[n]}), \mathcal{B})$, where $\mathcal{B}$ denotes the set of identified cheaters. Additionally, the parties run (input-independent and function-independent) Round 1 of π_{GOD}.

Round 3–4: If $S_i = \perp$, the parties in $\mathcal{P} \setminus \mathcal{B}$ run Round 2 and 3 of π_{GOD} computing $f^{\mathcal{B}}$ (f with the inputs of parties in $\mathcal{B}$ are hardcoded to default values) and output y as the outcome of π_{GOD}. Else, participate in $\mathsf{ARec}^{\lfloor n/2 \rfloor}$ with input S_i and output the outcome of $\mathsf{ARec}^{\lfloor n/2 \rfloor}$.

Figure 6.7: Robust MPC against boundary-admissible adversary

We state the formal theorem below.

Theorem 6.6 *Assuming the presence of a 2-round protocol π_{idua} realizing $\mathcal{F}_{\mathsf{idua}}$ (Figure 2.5) against malicious majority and a 3-round protocol π_{GOD} realizing $\mathcal{F}_{\mathsf{god}}$ in the presence of honest majority (with special property of Round 1 being function and input-independent), the 4-round MPC protocol $\pi^{\mathsf{bou}}_{\mathsf{GOD}}$ (Figure 6.7) satisfies:*

– *Correctness: computes the correct output.*

– *Security: realizes $\mathcal{F}_{\mathsf{god}}$ (Figure 2.4) against boundary-admissible $\mathcal{A}$*

Proof: Correctness of $\pi^{\mathsf{bou}}_{\mathsf{GOD}}$ follows directly from that of π_{idua}, π_{GOD} and $\mathsf{ARec}^{\lfloor n/2 \rfloor}$ (Lemma 6.13). We prove its security in Section 6.6.3.2. □

We conclude this section with a simplification to $\pi^{\mathsf{bou}}_{\mathsf{GOD}}$ that can be adopted if additional access to PKI is assumed. In such a case, parallelizing Round 1 of π_{GOD} with Round 1 of π_{idua} can be avoided and the 2-round honest-majority protocol of [108] achieving GOD assuming CRS and PKI setup can be used to instantiate π_{GOD} (which would be run in Rounds 3-4 of $\pi^{\mathsf{bou}}_{\mathsf{GOD}}$). Both our 4-round constructions with CRS (Figure 6.7) and its simplified variant with CRS and PKI are tight upper bounds, in light of the impossibility of Section 6.4.1 that holds in the presence of CRS and PKI.

6.5.3 Upper bound for Robust MPC: The single corruption case

Building upon the ideas of Section 6.5.2 and Section 6.3.3, a 3-round robust MPC $\pi^{\mathsf{bou},1}_{\mathsf{GOD}}$ against the special-case boundary-admissible adversary can be constructed as follows. Similar to $\pi^{\mathsf{bou}}_{\mathsf{GOD}}$, Round 1 and 2 involve running protocol π_{idua} realizing $\lfloor n/2 \rfloor$-authenticated secret-sharing of the function output. When π_{idua} does not result in abort, $\pi^{\mathsf{bou},1}_{\mathsf{GOD}}$ proceeds to reconstruction of output; identical to $\pi^{\mathsf{bou}}_{\mathsf{GOD}}$ and thereby terminating in 3 rounds. However, when π_{idua} results in output $\perp$, we exploit the advantage of atmost one malicious corruption by noting that once the single actively-corrupt party is expelled, the parties involved thereafter comprise only of the honest and purely passive parties. We adopt the idea of Section 6.3.3 and re-run Round 2 of π_{idua} among the remaining parties to compute the function output directly, with input of the expelled party substituted with default input. This step demands the function-delayed property of π_{idua} i.e Round 1 is independent of the function to be computed and the number of parties. In order to accommodate this re-run, two instances of Round 1 of π_{idua} are run in Round 1 of $\pi^{\mathsf{bou},1}_{\mathsf{GOD}}$. It is easy to see that robustness is ensured as π_{idua} is robust in the absence of actively-corrupt parties. Lastly, we point that similar to Section 6.3.3, we use the modified variant of the 2-round protocol of [93] to instantiate π_{idua} that is function-delayed and achieves identifiability. The formal description of $\pi^{\mathsf{bou},1}_{\mathsf{GOD}}$ appears in Figure 6.8. This upper bound is tight, following the impossibility of 2-round fair MPC (that holds for single malicious corruption) proven in Section

6.4.2 as GOD implies fairness. Assumption wise, $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$ relies on 2-round maliciously-secure OT in the common random/reference string model and NIZK (when π_{idua} is instantiated with function-delayed variant of the protocol of [93] satisfying identifiability).

Protocol $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$

Inputs: Party P_i has x_i for $i \in [n]$

Building Blocks: (a) 2-round protocol π_{idua} achieving identifiable abort against malicious majority (realizing functionality $\mathcal{F}_{\mathsf{idua}}$, refer Figure 2.5) and having function-delayed property; (b) Protocol $\mathsf{ARec}^{\lfloor n/2 \rfloor}$ for reconstructing an $\lfloor n/2 \rfloor$-authenticated-shared secret (Figure 6.6); (c) Function $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor}$ (Figure 6.5).

Output: $y = f(x_1 \dots x_n)$

Round 1: P_i does the following: Run 2 instances of Round 1 of π_{idua}, each using input x_i and independent randomness. Note that this round is independent of the function to be computed and the number of parties.

Round 2: P_i does the following: Run Round 2 of π_{idua} (based on first instance of Round 1 of π_{idua}) among $\mathcal{P}$ computing the function $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor} \diamond f$ using input x_i to obtain output $(S_i = (\mathsf{sh}_i, \{k_{ji}\}_{j\in[n]}, \{\mathsf{tag}_{ij}\}_{j\in[n]}), \mathcal{B})$, where $\mathcal{B}$ denotes the set of identified cheaters.

Round 3: If $S_i = \perp$, the parties in $\mathcal{P} \setminus \mathcal{B}$ run Round 2 of π_{idua} (based on second instance of Round 1 of π_{idua}) computing $f^{\mathcal{B}}$ (where the inputs of the party in $\mathcal{B}$ is hardcoded to default value) and output y as the outcome of this (second) instance of π_{idua}. Else, participate in $\mathsf{ARec}^{\lfloor n/2 \rfloor}$ with input S_i and output the outcome of $\mathsf{ARec}^{\lfloor n/2 \rfloor}$.

Figure 6.8: Robust MPC against special-case boundary-admissible adversary

We state the formal theorem below.

Theorem 6.7 *Assuming the presence of a 2-round protocol π_{idua} realizing functionality $\mathcal{F}_{\mathsf{idua}}$ (Figure 2.5) against malicious majority and having function-delayed property, the 3-round MPC protocol $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$ (Figure 6.8) satisfies:*

- *Correctness: computes the correct output.*
- *Security: realizes $\mathcal{F}_{\mathsf{god}}$ (Figure 2.4) against special-case boundary-admissible $\mathcal{A}$ with corruption parameters either $(t_a, t_p) = (1, \lfloor n/2 \rfloor)$ or $(t_a, t_p) = (0, n-1)$.*

Proof: Correctness of $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$ follows directly from correctness of π_{idua}, and correctness of $\mathsf{ARec}^{\lfloor n/2 \rfloor}$ (Lemma 6.13). We prove its security in Section 6.6.3.3. □

6.5.4 Upper bound for Fair MPC

The 4-round robust protocol $\pi^{\mathsf{bou}}_{\mathsf{GOD}}$ (Section 6.5.2) can be simplified as follows to yield a 3-round fair MPC protocol $\pi^{\mathsf{bou}}_{\mathsf{fn}}$. Similar to $\pi^{\mathsf{bou}}_{\mathsf{GOD}}$, Round 1 and 2 involve execution of π_{ua} (instantiated by [93, 35] in the CRS model) achieving unanimous abort against malicious-majority (identifiability is not needed) in order to compute $\lfloor n/2 \rfloor$-authenticated sharing of the output. If π_{ua} does not result in abort, the honest parties proceed to reconstruction of output in Round 3. Else, the honest parties simply output $\perp$. It is easy to check that fairness is preserved due to privacy of $\lfloor n/2 \rfloor$-authenticated secret-sharing (Lemma 6.13). Protocol $\pi^{\mathsf{bou}}_{\mathsf{fn}}$ appears in Figure 6.9 and is round-optimal, in view of the lower bound of Section 6.4.2. Assumption wise, $\pi^{\mathsf{bou}}_{\mathsf{fn}}$ relies on 2-round maliciously-secure OT in the common random/reference string model (when π_{ua} is instantiated with the protocols of [93, 35]).

Protocol $\pi^{\mathsf{bou}}_{\mathsf{fn}}$

Inputs: Party P_i has x_i for $i \in [n]$

Building Blocks: (a) 2-round protocol π_{ua} achieving security with unanimous abort against malicious majority (realizing functionality $\mathcal{F}_{\mathsf{ua}}$, refer Figure 2.2); (b) Protocol $\mathsf{ARec}^{\lfloor n/2 \rfloor}$ for reconstructing an $\lfloor n/2 \rfloor$-authenticated-shared secret (Figure 6.6); (c) Function $f^{\lfloor n/2 \rfloor}_{\mathsf{ASh}}$ (Figure 6.5).

Output: $y = f(x_1 \dots x_n)$ or $\perp$.

Round 1–2: The parties run π_{ua} computing the function $f^{\lfloor n/2 \rfloor}_{\mathsf{ASh}} \diamond f$ with input x_i to obtain output $(S_i = (\mathsf{sh}_i, \{k_{ji}\}_{j\in[n]}, \{\mathsf{tag}_{ij}\}_{j\in[n]}))$.

Round 3: If $S_i = \perp$, the parties output $\perp$. Else, participate in $\mathsf{ARec}^{\lfloor n/2 \rfloor}$ with input S_i and output the outcome of $\mathsf{ARec}^{\lfloor n/2 \rfloor}$.

Figure 6.9: Fair MPC against boundary-admissible adversary

We state the formal theorem below.

Theorem 6.8 *Assuming the presence of a 2-round protocol π_{ua} realizing functionality $\mathcal{F}_{\mathsf{ua}}$ (Figure 2.2) against malicious majority, the 3-round MPC protocol $\pi^{\mathsf{bou}}_{\mathsf{fn}}$ (Figure 6.9) satisfies:*

- *Correctness: computes the correct output.*
- *Security: realizes against (t_a, t_p) boundary-admissible $\mathcal{A}$ (1) $\mathcal{F}_{\mathsf{fair}}$ (Figure 2.3) when $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ (2) $\mathcal{F}_{\mathsf{god}}$ (Figure 2.4) when $(t_a, t_p) = (0, n-1)$.*

Proof: Correctness of $\pi^{\mathsf{bou}}_{\mathsf{fn}}$ follows directly from correctness of π_{ua} and the correctness of $\mathsf{ARec}^{\lfloor n/2 \rfloor}$ (Lemma 6.13). We prove its security in Section 6.6.3.4. □

6.6 Security Proofs

6.6.1 Proofs for Upper Bounds for Dynamic Corruption

6.6.1.1 Ideal Functionality $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$

We formally define the ideal functionality computing the $(n-2, \lfloor \frac{n}{2} \rfloor)$-levelled sharing (Definition 6.3) of the output $y = f(x_1, \ldots x_n)$ securely with unanimous abort in Figure 6.10. This ideal functionality is identical to $\mathcal{F}_{\mathsf{ua}}$, with the only difference being that the relevant function computed is $f_{\mathsf{LSh}}^{n-2,\lfloor \frac{n}{2} \rfloor} \diamond f$. Refer Figure 6.1 for the description of $f_{\mathsf{LSh}}^{n-2,\lfloor \frac{n}{2} \rfloor}$.

Functionality $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$

Input: Receive message $\big((\mathcal{D}, \mathcal{E}), \mathsf{Input}, \{x_i\}_{i \in \mathcal{E}}\big)$ from $\mathcal{A}$. Next, do the following: if $(*, \mathsf{Input}, *)$ message was received from P_i, then ignore. Otherwise record it internally. If x_i is outside of the domain for P_i ($P_i \in$), consider $x_i = \texttt{abort}$.

Output to adversary: If there exists $i \in [n]$ such that $x_i = \texttt{abort}$, send $(\mathsf{sid}, \texttt{Output}, \perp)$ to all the parties. Else, compute $y = f(x_1 \ldots x_n)$ and compute $(L_1, \ldots L_n) = f_{\mathsf{LSh}}^{n-2,\lfloor \frac{n}{2} \rfloor}(y)$. Send $(\mathsf{sid}, \texttt{Output}, \{L_i\}_{i \in \mathcal{E}})$ to $\mathcal{A}$.

Output to honest parties: Receive either `continue` or `abort` from $\mathcal{A}$. In case of `continue`, send L_i to each honest P_i, whereas in case of `abort` send $\perp$ to all honest parties.

Figure 6.10: Ideal Functionality $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$

6.6.1.2 Security Proof of $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$ (Theorem 6.2)

We analyze the protocol $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$ in a $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$-hybrid model where the parties have access to a trusted party $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$ (Figure 6.10). Let $\mathcal{A}$ be a dynamic adversary with threshold (t_a, t_p) that controls t_p parties passively and upto t_a among them actively in the $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$-hybrid model execution of $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$. We describe a simulator $\mathcal{S}_{\mathsf{fn}}^{\mathsf{dyn}}$, running an ideal-world evaluation of the functionality $\mathcal{F}_{\mathsf{fair}}$ (refer Figure 2.3) computing f whose behaviour simulates the behaviour of $\mathcal{A}$ in Figure 6.11.

Simulator $\mathcal{S}_{\mathsf{fn}}^{\mathsf{dyn}}$

Recall that $\mathcal{D}(\mathcal{E})$ denotes the set of actively (passively) corrupt parties where $\mathcal{D} \subseteq \mathcal{E}$. Here $|\mathcal{D}| = t_a$ and $|\mathcal{E}| = t_p$. Let $\mathcal{H}$ and $\mathcal{C}$ denote the set of *indices* of honest parties (in $\mathcal{P} \setminus \mathcal{E}$) and parties in $\mathcal{E}$ respectively. $\mathcal{S}_{\mathsf{fn}}^{\mathsf{dyn}}$ does the following:

– **Interaction with $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor\frac{n}{2}\rfloor)}$:** Receive $((\mathcal{D},\mathcal{E}),\mathsf{Input},\{x_j\}_{j\in\mathcal{C}})$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor\frac{n}{2}\rfloor)}$. Compute $t_a=|\mathcal{D}|, t_p=|\mathcal{E}|$. If for any $P_j\in\mathcal{D}$, x_j is outside of domain of input, send $\perp$ as output of $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor\frac{n}{2}\rfloor)}$ to $\mathcal{A}$ and send $\perp$ as input to $\mathcal{F}_{\mathsf{fair}}$ on behalf of $\mathcal{A}$. Else continue.

– **Output of $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor\frac{n}{2}\rfloor)}$ to $\mathcal{A}$:** We have two cases.

- If $t_a=0$, invoke $\mathcal{F}_{\mathsf{fair}}$ on behalf of $\mathcal{A}$ with $\{x_j\}_{j\in\mathcal{C}}$ to receive an output value y in return. Compute $(L_1,\ldots L_n)=f_{\mathsf{LSh}}^{n-2,\lfloor\frac{n}{2}\rfloor}(y)$ (Figure 6.1) and return $\{L_j\}_{j\in\mathcal{C}}$ to $\mathcal{A}$ as output from $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor\frac{n}{2}\rfloor)}$.
- Else, do the following: Let $\alpha'=n-2$ and $\beta'=\lfloor n/2\rfloor$. For $j\in\mathcal{C}$, return $L_j=\left(\{s_{ij},o_{ij}\}_{i\in[\alpha',\beta']},\{c_{il}\}_{i\in[\alpha',\beta'],l\in[n]}\right)$ where s_{ij} are randomly chosen, $(c_{ij},o_{ij})\leftarrow\mathsf{eCom}(s_{ij};r_{ij})$ computed as per protocol specifications and $\{c_{il}\}_{i\in[\alpha',\beta'],l\in\mathcal{H}}$ are computed as commitments on dummy values, say involving $\{s'_{il},o'_{il}\}_{i\in[\alpha',\beta'],l\in\mathcal{H}}$.

– **Completing Simulation of Round 1 - 2:** If $\mathcal{A}$ invokes $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor\frac{n}{2}\rfloor)}$ with `abort`, invoke $\mathcal{F}_{\mathsf{fair}}$ with input $\perp$ on behalf of $\mathcal{A}$ and output $\perp$ on behalf of honest parties.

Note: Recall that in Round r ($r\in[3,\lceil n/2\rceil+1]$), summand s_{n-r+1} is attempted to be reconstructed (in Round $r-2$ of $\mathsf{LRec}^{n-2,\lfloor\frac{n}{2}\rfloor}$).

– **Round 3 to Round $(\mathbf{n-t_p})$:** $\mathcal{S}_{\mathsf{fn}}^{\mathsf{dyn}}$ does the following in Round r', where $r'=[3,n-t_p]$

- Let $i=n-r'+1$. Send $\{s'_{il},o'_{il}\}_{l\in\mathcal{H}}$ on behalf of honest parties and receive $\{s'_{ij},o'_{ij}\}_{j\in\mathcal{C}}$ from $\mathcal{A}$.
- Initialize $\mathcal{V}_i=\mathcal{P}\backslash$. Add $P_j\in$ to $\mathcal{V}_i$ if P_j sends $(s'_{ij},o'_{ij})=(s_{ij},o_{ij})$ (consistent with L_j returned as output of $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor\frac{n}{2}\rfloor)}$ to P_j). If $|\mathcal{V}_i|<i+1$, then abort and invoke $\mathcal{F}_{\mathsf{fair}}$ with input $\perp$ on behalf of $\mathcal{A}$; thereby completing simulation. Else, continue to $r'=r'+1$.

– **Round $(\mathbf{n-t_p+1})$:** This round involves reconstruction of summand s_{t_p}. $\mathcal{S}_{\mathsf{fn}}^{\mathsf{dyn}}$ does the following:

- Invoke $\mathcal{F}_{\mathsf{fair}}$ on behalf of $\mathcal{A}$ with $\{x_j\}_{j\in\mathcal{C}}$ to receive output y.
- Note that reconstruction of summands $s_{t_p+1},\ldots,s_{n-2}$ has been completed and the summands s_i, where $i\in[\lfloor n/2\rfloor,\ldots,t_p-1]$ is already fully determined by $\{s_{ij}\}_{j\in\mathcal{C}}$ returned as output of $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor\frac{n}{2}\rfloor)}$ to $\mathcal{A}$. Compute $s_{t_p}=y-\sum_{i=\lfloor n/2\rfloor}^{t_p-1}s_i-\sum_{i=t_p+1}^{n-2}s_i$.
- Let $\mu=t_p$. Interpolate a μ-degree polynomial $g_\mu(x)$ satisfying $g_\mu(0)=s_\mu$ and $g_\mu(j)=s_{\mu j}$ for $j\in\mathcal{C}$. Let $s_{\mu l}=g_\mu(l)$ for $l\in\mathcal{H}$. Compute $o_{\mu l}\leftarrow\mathsf{Equiv}(c_{\mu l},(s'_{\mu l},o'_{\mu l}),s_{\mu l},t)$ (Section 2.4.2.1). Broadcast $(s_{\mu l},o_{\mu l})$ on behalf of P_l, $l\in\mathcal{H}$.

– **Round $(\mathbf{n-t_p+2})$ to Round $(\lceil \mathbf{n/2} \rceil + \mathbf{1})$:** In Round r' ($r' \in [n-t_p+2, \lceil n/2 \rceil + 1]$), broadcast (s'_{il}, o'_{il}) on behalf of P_l, $l \in \mathcal{H}$; where $i = n - r' + 1$.

Figure 6.11: Simulator $\mathcal{S}_{\mathsf{fn}}^{\mathsf{dyn}}$

At a high-level, the simulation is divided into 4 parts: Rounds 1-2, Rounds 3 to $n-t_p$, Round $n-t_p+1$ and finally Rounds $n-t_p+1$ to Round $\lceil n/2 \rceil + 1$. In order to complete the proof, we argue how each of them maintain that the view of $\mathcal{A}$ is the ideal world is indistinguishable from its view in $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$-hybrid model execution of $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$ (hybrid-world):

Rounds 1 - 2. It is straightforward to check that the view of $\mathcal{A}$ in the real and hybrid-world are indistinguishable. Note that incase of no active corruptions, $\mathcal{F}_{\mathsf{fair}}$ is invoked on behalf of $\mathcal{A}$ to get the output directly which is consistent with the hybrid-world, where the output of f can be deduced by $\mathcal{A}$ corrupting $t_p = n-1$ parties from the output of $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$. Furthermore, if $\mathcal{A}$ invokes $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$ with `abort`, it must hold that $t_a \geq 1$ implying $t_p < n-1$. In this case $\mathcal{F}_{\mathsf{fair}}$ is invoked with $\perp$, which is consistent with the hybrid-world where $\mathcal{A}$ has no information about s_{n-2} and consequently the output. This follows directly from the property of Shamir-Sharing and hiding property of eNICOM.

Rounds 3 to Round $\mathbf{n-t_p}$. Note that these rounds involve only reconstruction of summands $s_{n-2} \ldots s_{t_p+1}$. Indistinguishability follows from the fact that in both ideal and hybrid-world, $\mathcal{A}$ corrupting upto t_p number of parties has no information regarding the summand s_{t_p} and consequently the output y. This can be inferred from the property of Shamir-sharing of s_{t_p} with threshold t_p and the hiding property of eNICOM. Next, we observe that the only difference in the ideal and the hybrid-model is the following: In the hybrid-model, the share of a party P_j, say s_{ij} ($i = [n-2, t_p+1]$) is discarded during $\mathsf{LRec}^{n-2,\lfloor n/2 \rfloor}()$ if the corresponding commitment c_{ij} (output from $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$) does not open successfully using the given opening o'_{ij} obtained from P_j. However, in the ideal world, the share of P_j ($P_j \in$) is discarded if P_j does not send (s_{ij}, o_{ij}), same as output from $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$. It follows from the binding property of the equivocal commitment eNICOM that P_j will not be able to send $(s'_{ij}, o'_{ij}) \neq (s_{ij}, o_{ij})$ such that $\mathsf{eOpen}(\mathsf{epp}, c_{ij}, o'_{ij}) = s'_{ij}$, except with negligible probability. Thus, indistinguishability holds.

Round $\mathbf{n-t_p+1}$. This constitutes the crux of the simulation. We observe that if reconstructions of summands upto s_{t_p+1} were successful, in the hybrid-world, $\mathcal{A}$ can deduce the output in Round $n-t_p+1$ involving reconstruction of s_{t_p} (Summands $s_{\lfloor n/2 \rfloor}, \ldots, s_{t_p-1}$ are already fully determined by output of $\mathcal{A}$ received from $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$). To maintain indistiguishability, $\mathcal{S}_{\mathsf{fn}}^{\mathsf{dyn}}$ invokes $\mathcal{F}_{\mathsf{fair}}$ to obtain output y and sets s_{t_p} accordingly so that

$\sum_{i=\lfloor n/2 \rfloor}^{n-2} s_i = y$. The only difference between the ideal and hybrid-world is the following: In the hybrid world, for $i = t_p$, the commitments $\{c_{il}\}_{l \in \mathcal{H}}$ correspond to $\{s_{il}, o_{il}\}$ computed as per output y. However in the ideal world, $\{c_{il}\}_{l \in \mathcal{H}}$ were commitments on dummy values that were later equivocated to appropriate values of shares of honest parties as per the computed s_{t_p} (set such that the summands add upto y). Indistinguishability follows from the properties of equivocal commitment schemes (Section 2.4.2.1).

Round $n - t_p + 2$ to Round $(\lceil n/2 \rceil + 1)$. It is easy to check that the view of $\mathcal{A}$ is identical in the ideal and hybrid-world.

This completes the proof of Theorem 6.2.

6.6.2 Security Proof of $\pi^{\mathsf{dyn}}_{\mathsf{GOD}}$ (Theorem 6.3)

Let $\mathcal{A}$ be a dynamic-admissible adversary with threshold (t_a, t_p) that controls t_p parties passively and upto t_a among them actively during an execution of $\pi^{\mathsf{dyn}}_{\mathsf{GOD}}$. We prove Theorem 6.3 by describing a simulator $\mathcal{S}^{\mathsf{dyn}}_{\mathsf{GOD}}$, running an ideal-world evaluation of the functionality $\mathcal{F}_{\mathsf{god}}$ (refer Figure 2.4) computing f whose behaviour simulates the behaviour of $\mathcal{A}$ in Figure 6.12. $\mathcal{S}^{\mathsf{dyn}}_{\mathsf{GOD}}$ invokes the simulator of the subprotocol π_{idua}, say $\mathcal{S}_{\pi_{\mathsf{idua}}}$ (running an ideal-world evaluation of functionality $\mathcal{F}_{\mathsf{idua}}$, refer Figure 2.5).

Simulator $\mathcal{S}^{\mathsf{dyn}}_{\mathsf{GOD}}$

Let $\mathcal{D}(\mathcal{E})$ denote the set of actively (passively) corrupt parties where $\mathcal{D} \subseteq \mathcal{E}$. Here $|\mathcal{D}| = t_a$ and $|\mathcal{E}| = t_p$. Let $\mathcal{H}$ and $\mathcal{C}$ denote the set of *indices* of honest parties (in $\mathcal{P} \setminus \mathcal{E}$) and parties in $\mathcal{E}$ respectively. The following steps are carried out by $\mathcal{S}^{\mathsf{dyn}}_{\mathsf{GOD}}$:

- **Step 1:** For $k = 1$ to $\lceil n/2 \rceil$, let $\mathsf{m}_i^{1,k} \leftarrow \mathcal{S}_{\pi_{\mathsf{idua}}}(r_i^k)$ $(i \in \mathcal{H})$ correspond to the Round 1 message of P_i for the kth instance obtained by invoking $\mathcal{S}_{\pi_{\mathsf{idua}}}$ with fresh independent randomness r_i^k. Note that this message is independent of the function to be computed by π_{idua} and the number of parties. Send $\{\mathsf{m}_i^{1,k}\}_{k \in [\lceil n/2 \rceil]}$ on behalf of P_i to $\mathcal{A}$. Receive $\{\mathsf{m}_j^{1,k}\}_{k \in [\lceil n/2 \rceil]}$ sent by P_j $(P_j \in \mathcal{E})$.

- **Step 2:** Initialize $k = 1$, $\mathcal{L} = \mathcal{P}$, $\mathcal{C} = \emptyset$, $\mathfrak{n} = n$. Let $f^{\mathcal{C}}$ denote the function same as f except with default inputs hardcoded for parties in $\mathcal{C}$. Send $\mathsf{m}_j^{1,k}$ for each $P_j \in \mathcal{L} \cap \mathcal{E}$ to $\mathcal{S}_{\pi_{\mathsf{idua}}}$. When $\mathcal{S}_{\pi_{\mathsf{idua}}}$ returns the extracted input $\{x_j^k\}_{P_j \in \mathcal{L} \cap \mathcal{E}}$ to invoke its ideal functionality $\mathcal{F}_{\mathsf{idua}}$ on behalf of $\mathcal{A}$, if $x_j = \perp$ for $P_j \in \mathcal{L} \cap \mathcal{D}$, return $(\perp, \mathcal{B} = P_j)$ as output of $\mathcal{F}_{\mathsf{idua}}$.

Step 2.1: If $\mathfrak{n} = 1, 2$, invoke $\mathcal{F}_{\mathsf{god}}$ with $\{x_j^k\}_{P_j \in \mathcal{L} \cap \mathcal{E}}$ on behalf of corrupt parties that are alive and default inputs on behalf of identified actively corrupt parties in $\mathcal{C}$. Receive an output value y in return, which is forwarded to $\mathcal{S}_{\pi_{\mathsf{idua}}}$ as response from $\mathcal{F}_{\mathsf{idua}}$. Let $\mathsf{m}_i^{2,k} \leftarrow \mathcal{S}_{\pi_{\mathsf{idua}}}(\mathsf{T}_1^k, y, f^{\mathcal{C}}, \mathcal{L}; r_i^k)$ $(i \in \mathcal{H})$ correspond to the Round 2 message of P_i obtained by invoking $\mathcal{S}_{\pi_{\mathsf{idua}}}$ with P_i's ran-

domness r_i^k, transcript of Round 1 (kth instance) i.e $\mathsf{T}_1^k = \{\mathsf{m}_j^{1,k}\}_{j\in\mathcal{C}}$, output y, $f^{\mathcal{C}}$ as the function to be computed and $\mathcal{L}$ as the parties involved in computation. Send $\mathsf{m}_i^{2,k}$ on behalf of P_i to $\mathcal{A}$. Receive $\mathsf{m}_j^{2,k}$ as Round 2 message of π_{idua} sent by P_j ($P_j \in \mathcal{E}$) and send it to $\mathcal{S}_{\pi_{\mathsf{idua}}}$ on behalf of P_j. This completes the simulation.

Step 2.2: Else, we have two cases (similar to $\mathcal{S}_{\mathsf{fn}}^{\mathsf{dyn}}$). Let $\alpha' = \mathfrak{n} - 2$ and $\beta' = \lfloor \mathfrak{n}/2 \rfloor$.

- If $t_a = 0$, invoke $\mathcal{F}_{\mathsf{god}}$ with $\{x_j^k\}_{P_j\in\mathcal{L}\cap\mathcal{E}}$ on behalf of corrupt parties that are alive and default inputs on behalf of identified actively corrupt parties in $\mathcal{C}$. Receive an output value y in return. Compute $(L_1, \ldots L_q) = f_{\mathsf{LSh}}^{\alpha',\beta'}(y)$ (Figure 6.1) among parties in $\mathcal{L}$ (where $q = |\mathcal{L}|$) and return $y' = \{L_j\}$ i.e the set of L_js for all $P_j \in \mathcal{L}\cap\mathcal{E}$ as output of $\mathcal{F}_{\mathsf{idua}}$ to $\mathcal{S}_{\pi_{\mathsf{idua}}}$.
- Else, for $P_j \in \mathcal{E}$, set $L_j = \left(\{s_{ij}, o_{ij}\}_{i\in[\alpha',\beta']}, \{c_{il}\}_{i\in[\alpha',\beta'],P_l\in\mathcal{L}}\right)$ where s_{ij} ($i \in [\alpha', \beta']$) are randomly chosen and $(c_{ij}, o_{ij}) \leftarrow \mathsf{eCom}(s_{ij}; r_{ij})$ computed as per protocol specifications. $\{c_{il}\}_{i\in[\alpha',\beta'],l\in\mathcal{H}}$ are computed as commitments on dummy values, say involving $\{s'_{il}, o'_{il}\}_{i\in[\alpha',\beta'],l\in\mathcal{H}}$. Return $y' = \{L_j\}$ i.e the set of L_js for all $P_j \in \mathcal{L}\cap\mathcal{E}$ as response to $\mathcal{S}_{\pi_{\mathsf{idua}}}$ from $\mathcal{F}_{\mathsf{idua}}$.
- Let $\mathsf{m}_i^{2,k} \leftarrow \mathcal{S}_{\pi_{\mathsf{idua}}}(\mathsf{T}_1^k, y', f_{\mathsf{LSh}}^{\alpha',\beta'} \diamond f^{\mathcal{C}}, \mathcal{L}; r_i^k)$ ($i \in \mathcal{H}$) correspond to the Round 2 message of P_i obtained by invoking $\mathcal{S}_{\pi_{\mathsf{idua}}}$ with P_i's randomness r_i^k, transcript of Round 1 i.e $\mathsf{T}_1^k = \{\mathsf{m}_j^{1,k}\}_{j\in\mathcal{C}}$, output y', function $f_{\mathsf{LSh}}^{\alpha',\beta'} \diamond f^{\mathcal{C}}$ i.e (α', β') levelled-Sharing of output of $f^{\mathcal{C}}$ as the function to be computed and $\mathcal{L}$ as the parties involved in computation. Send $\mathsf{m}_i^{2,k}$ on behalf of P_i to $\mathcal{A}$. Receive $\mathsf{m}_j^{2,k}$ as Round 2 message of π_{idua} sent by P_j ($P_j \in \mathcal{E}$) and send it to $\mathcal{S}_{\pi_{\mathsf{idua}}}$ on behalf of P_j.

There are 2 cases based on whether $\mathcal{A}$ aborts the computation of π_{idua}.

- If $\mathcal{S}_{\pi_{\mathsf{idua}}}$ invokes $\mathcal{F}_{\mathsf{idua}}$ with $(\mathtt{abort}, \mathcal{B})$ with $\mathcal{B} \subseteq \mathcal{D}$ or $(\perp, \mathcal{B})$ had been returned as output of $\mathcal{F}_{\mathsf{idua}}$ to $\mathcal{A}$, update $\mathcal{C} = \mathcal{C}\cup\mathcal{B}$, $\mathcal{L} = \mathcal{L}\setminus\mathcal{B}$, $\mathcal{D} = \mathcal{D}\setminus\mathcal{B}$, $\mathfrak{n} = \mathfrak{n} - 2|\mathcal{B}|$, $k = k+1$, $t_a = t_a - |\mathcal{B}|$ and repeat this simulation of step 2 using updated value of $\mathfrak{n}, k$ and the updated sets.
- Else, if $\mathcal{S}_{\pi_{\mathsf{idua}}}$ invokes $\mathcal{F}_{\mathsf{idua}}$ with $\mathtt{continue}$, run the following steps to simulate $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor\frac{\mathfrak{n}}{2}\rfloor}$ (similar to analogous steps in $\mathcal{S}_{\mathsf{fn}}^{\mathsf{dyn}}$). Recall that in Round r of $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor\frac{\mathfrak{n}}{2}\rfloor}$ ($r \in [1, \lceil\mathfrak{n}/2\rceil - 1]$), summand $s_{\mathfrak{n}-r-1}$ is attempted to be reconstructed. $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{dyn}}$ does the following in Round r of $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor\frac{\mathfrak{n}}{2}\rfloor}$:

  1. If $r \leq \mathfrak{n} - t_p - 2$: Let $i = \mathfrak{n} - r - 1$. Send $\{s'_{il}, o'_{il}\}_{l\in\mathcal{H}}$ on behalf of honest parties and receive (s'_{ij}, o'_{ij}) from each $P_j \in \mathcal{L}\cap\mathcal{E}$. Initialize $\mathcal{V}_i = \mathcal{L}\setminus\mathcal{D}$. Add $P_j \in \mathcal{V}_i$ if $P_j \in \mathcal{D}$ sends $(s'_{ij}, o'_{ij}) = (s_{ij}, o_{ij})$ (consistent with L_j returned as output of $\mathcal{F}_{\mathsf{idua}}$ to P_j). If $|\mathcal{V}_i| < i+1$, then let $\mathcal{B} = \mathcal{L}\setminus\mathcal{V}_i$. Update $\mathcal{C} = \mathcal{C}\cup\mathcal{B}$, $\mathcal{L} = \mathcal{L}\setminus\mathcal{B}$, $\mathcal{D} = \mathcal{D}\setminus\mathcal{B}$, $\mathfrak{n} = \mathfrak{n} - 2|\mathcal{B}|$, $k = k+1$, $t_a = t_a - |\mathcal{B}|$, $t_p = t_p - |\mathcal{B}|$ and repeat the simulation of step 2 using these updated values.

2. If $r = \mathfrak{n} - t_p - 1$: This round involves reconstruction of summand s_{t_p}. $\mathcal{S}^{\mathsf{dyn}}_{\mathsf{GOD}}$ does the following:
   - Invoke $\mathcal{F}_{\mathsf{god}}$ with $\{x_j^k\}_{P_j \in \mathcal{L} \cap \mathcal{E}}$ on behalf of corrupt parties that are alive and default inputs on behalf of identified actively corrupt parties in $\mathcal{C}$. Receive output value y in return.
   - Note that reconstruction of summands $s_{t_p+1} \ldots s_{\mathfrak{n}-2}$ has been completed and the summands s_i, where $i \in [\lfloor \mathfrak{n}/2 \rfloor, \ldots t_p - 1]$ is already fully determined by values returned as output of $\mathcal{F}_{\mathsf{idua}}$. Compute $s_{t_p} = y - \sum_{i=\lfloor \mathfrak{n}/2 \rfloor}^{t_p-1} s_i - \sum_{i=t_p+1}^{\mathfrak{n}-2} s_i$.
   - Let $\mu = t_p$. Interpolate a μ-degree polynomial $g_\mu(x)$ satisfying $g_\mu(0) = s_\mu$ and $g_\mu(j) = s_{\mu j}$ for $P_j \in \mathcal{E} \cap \mathcal{L}$. Let $s_{\mu l} = g_\mu(l)$ for $l \in \mathcal{H}$. Compute $o_{\mu l} \leftarrow \mathsf{Equiv}(c_{\mu l}, (s'_{\mu l}, o'_{\mu l}), s_{\mu l}, t)$. Send $(s_{\mu l}, o_{\mu l})$ on behalf of P_l, $l \in \mathcal{H}$.
3. If $r \in [(\mathfrak{n} - t_p), \lceil \mathfrak{n}/2 \rceil - 1]$: Send (s'_{il}, o'_{il}) on behalf of P_l, $l \in \mathcal{H}$, where $i = \mathfrak{n} - r - 1$.

Figure 6.12: Simulator $\mathcal{S}^{\mathsf{dyn}}_{\mathsf{GOD}}$

The argument to show that the view of $\mathcal{A}$ in the ideal world is indistinguishable from its view in the real-world i.e during an execution of $\pi^{\mathsf{dyn}}_{\mathsf{GOD}}$ is an extension of the security argument for $\pi^{\mathsf{dyn}}_{\mathsf{fn}}$. Firstly, it is easy to check that the simulation proceeds identical to that of $\pi^{\mathsf{dyn}}_{\mathsf{fn}}$ incase neither π_{idua} nor any invocation of $\mathsf{LRec}^{\alpha',\beta'}()$ fails with the only difference that the messages of the honest parties in Round 1, 2 are simulated by invoking $\mathcal{S}_{\pi_{\mathsf{idua}}}$ (as opposed to $\mathcal{F}_{\mathsf{ua}}^{(n-2,\lfloor \frac{n}{2} \rfloor)}$-hybrid model analysis in $\pi^{\mathsf{dyn}}_{\mathsf{fn}}$) and involve multiple Round 1 instances of π_{idua} whose simulation is indistinguishable to the real world. It thus follows directly from security of π_{idua} that the view of $\mathcal{A}$ in the ideal world is indistinguishable from its view in the real-world in such cases.

Next, we note that during execution of π_{idua}, the output from its ideal functionality $\mathcal{F}_{\mathsf{idua}}$ can be appropriately simulated based on whether there are any active parties or not, similar to $\mathcal{S}^{\mathsf{dyn}}_{\mathsf{fn}}$ (Figure 6.11). If π_{idua} aborts by exposing (atleast one) cheater adding to set $\mathcal{C}$, then the simulation of Round 2 of π_{idua} is re-run wrt set of parties $\mathcal{L} = \mathcal{L} \setminus \mathcal{C}$ and updated $\mathfrak{n}$ and the modified function $f_{\mathsf{LSh}}^{\mathfrak{n}-2,\lfloor \frac{\mathfrak{n}}{2} \rfloor} \diamond f^{\mathcal{C}}$ computing levelled-sharing of output of $f^{\mathcal{C}}$. When π_{idua} succeeds, the simulation of the steps of $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor \frac{\mathfrak{n}}{2} \rfloor}()$ is identical to $\mathcal{S}^{\mathsf{dyn}}_{\mathsf{fn}}$ with the following difference: If $\mathsf{LRec}^{\mathfrak{n}-2,\lfloor \frac{\mathfrak{n}}{2} \rfloor}()$ returns $\perp$, the identified set of actively corrupt parties are eliminated from the computation and the simulation of Step 2 is re-run wrt the remaining parties and the modified function to be computed. To give better insight to the security of protocol, we emphasize that $\mathcal{S}^{\mathsf{dyn}}_{\mathsf{GOD}}$ invokes $\mathcal{F}_{\mathsf{god}}$ only once: Either when computation involves no active parties ($t_a = 0$ or $\mathfrak{n} = 1, 2$) or when the gradual reconstruction of levelled-shared output proceeds without failure until summand s_{t_p+1}. The latter case is consistent with the real-world where $\mathcal{A}$ corrupting t_p parties would obtain complete information about the output if he does

not disrupt reconstruction upto summand s_{t_p+1}. Thus, it is evident that $\mathcal{A}$ gets only unique output as $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{dyn}}$ invokes $\mathcal{F}_{\mathsf{god}}$ only once while maintaining throughout the execution that view of $\mathcal{A}$ in the ideal world is indistinguishable from its view in the real-world. This completes the sketch of the simulation.

It is now easy to check that the formal security proof of $\pi_{\mathsf{GOD}}^{\mathsf{dyn}}$ can be derived in a straightforward manner from the security proof of $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$ (Section 6.6.1.2). We can thus conclude that it follows directly from the security proof arguments of $\pi_{\mathsf{fn}}^{\mathsf{dyn}}$ and the security of π_{idua} that the simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{dyn}}$ outputs a view indistinguishable to the view of $\mathcal{A}$ in $\pi_{\mathsf{GOD}}^{\mathsf{dyn}}$.

6.6.3 Proofs of Upper Bounds for Boundary Corruption

6.6.3.1 Ideal Functionality $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$

We formally define the ideal functionality $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$ computing the authenticated sharing of the output $y = f(x_1, \ldots x_n)$ securely with abort in Figure 6.13. This ideal functionality is identical to $\mathcal{F}_{\mathsf{ua}}$, with the only difference being that the relevant function computed is $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor} \diamond f$. Refer Figure 6.5 for the description of $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor}$.

Functionality $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$

Input: On message $(\mathsf{sid}, \mathsf{Input}, x_i)$ from a party P_i $(i \in [n])$, do the following: if $(\mathsf{sid}, \mathsf{Input}, *)$ message was received from P_i, then ignore. Otherwise record it internally. If x_i is outside of the domain for P_i $(i \in [n])$, consider $x_i = \texttt{abort}$.

Output to adversary: If there exists $i \in [n]$ such that $x_i = \texttt{abort}$, send $(\mathsf{sid}, \texttt{Output}, \perp)$ to all the parties. Else, compute $y = f(x_1 \ldots x_n)$ and $(S_1, \ldots S_n) = f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor}(y)$. Send $(\mathsf{sid}, \texttt{Output}, \{S_i\}_{i \in \mathcal{C}})$ to the adversary, where $\mathcal{C}$ denotes the set of parties controlled by the adversary.

Output to honest parties: Receive either `continue` or `abort` from adversary. In case of `continue`, send S_i to each honest P_i, whereas in case of `abort` send $\perp$ to all honest parties.

Figure 6.13: Ideal Functionality $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$

Similar to the above, functionality $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$ can be defined identical to $\mathcal{F}_{\mathsf{idua}}$, with the only difference being that the relevant function computed is $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor} \diamond f$.

6.6.3.2 Proof of Security of $\pi_{\mathsf{GOD}}^{\mathsf{bou}}$ (Theorem 6.6)

We prove Theorem 6.6 by presenting two separate simulators $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh}}$ and $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$ for the case of pure passive corruption $(t_a, t_p) = (0, n-1)$ and $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ involving active corruptions respectively. The protocol $\pi_{\mathsf{GOD}}^{\mathsf{bou}}$ is analyzed in a $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$- hybrid model where the

parties have access to a trusted party computing $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$ (as defined in the previous section). Additionally, the simulator of the subprotocol π_{GOD}, say $\mathcal{S}_{\pi_{\mathsf{GOD}}}$ is also invoked.

Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh}}$ wrt $(t_a, t_p) = (0, n-1)$: Let $\mathcal{A}$ be a boundary-admissible adversary with parameters $(t_a, t_p) = (0, n-1)$ in the $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$-hybrid model execution of $\pi_{\mathsf{GOD}}^{\mathsf{bou}}$ (hybrid-world). The simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh}}$, running an ideal-world evaluation of the functionality $\mathcal{F}_{\mathsf{god}}$ (refer Figure 2.4) computing f whose behaviour simulates the behaviour of $\mathcal{A}$ is described in Figure 6.14. It is straightforward to see that the view of $\mathcal{A}$ in the ideal world is indistinguishable from the view of $\mathcal{A}$ in the $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$-hybrid model execution of $\pi_{\mathsf{GOD}}^{\mathsf{bou}}$. The only difference is that in the ideal world, Round 1 of $\pi_{\mathsf{GOD}}^{\mathsf{bou}}$ is obtained via $\mathcal{S}_{\pi_{\mathsf{GOD}}}$, whose simulation is independent of the parties' inputs. We can thus conclude that $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh}}$ outputs a view indistinguishable to the view of $\mathcal{A}$ in the hybrid-world.

Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh}}$

Let $\mathcal{C} \subset [n]$ and $\mathcal{H}$ be the set of indices of corrupt and honest parties respectively. The following steps are carried out by $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh}}$:

- *Simulation of Round 1 of π_{GOD}:* Let $\mathsf{m}_i^1 \leftarrow \mathcal{S}_{\pi_{\mathsf{GOD}}}(r_i)$ $(i \in \mathcal{H})$ correspond to the Round 1 message of P_i obtained by invoking $\mathcal{S}_{\pi_{\mathsf{GOD}}}$ with P_i's randomness r_i. Recall that this step is independent of parties' inputs. Send m_i^1 on behalf of P_i to $\mathcal{A}$. Receive m_j^1 as Round 1 message of π_{GOD} sent by P_j $(j \in \mathcal{C})$.
- *Invoking $\mathcal{F}_{\mathsf{god}}$:* Receive $\{x_i\}_{i\in\mathcal{C}}$ corresponding to the parties controlled by passive adversary $\mathcal{A}$. Invoke $\mathcal{F}_{\mathsf{god}}$ on behalf of $\mathcal{A}$ with $\{x_i\}_{i\in\mathcal{C}}$ to receive an output value y in return.
- *Output of $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$ to $\mathcal{A}$:* Compute the authenticated secret-sharing of value y with threshold $\lfloor n/2 \rfloor$ (Figure 6.5) as $(S_1 \dots S_n) = f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor}(y)$ and send $S_j = \left(\mathsf{sh}_j, \{k_{ij}\}_{i\in[n]}, \{\mathsf{tag}_{ji}\}_{i\in[n]}\right)$ as output of $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$ to P_j $(j \in \mathcal{C})$.
- *Round 3:* For each $i \in \mathcal{H}$, send $(\mathsf{sh}_i, \mathsf{tag}_{ij})$ $(j \neq i)$ on behalf of P_i.

Figure 6.14: Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh}}$

Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$

Let $\mathcal{C} \subset [n]$ and $\mathcal{H} = [n] \setminus \mathcal{C}$ be the set of indices of the parties controlled by adversary and the honest parties respectively. The following steps are carried out by $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$:

- *Simulation of Round 1 of π_{GOD}:* Let $\mathsf{m}_i^1 \leftarrow \mathcal{S}_{\pi_{\mathsf{GOD}}}(r_i)$ $(i \in \mathcal{H})$ correspond to the Round 1 message of P_i obtained by invoking $\mathcal{S}_{\pi_{\mathsf{GOD}}}$ with P_i's randomness r_i. Recall that this step is independent

of parties' inputs. Send m_i^1 on behalf of P_i to $\mathcal{A}$. Receive m_j^1 as Round 1 message of π_{GOD} sent by P_j ($j \in \mathcal{C}$).

– *Interaction of $\mathcal{A}$ with $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$:* Receive $\{x_i\}_{i\in\mathcal{C}}$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$. If for any $i \in \mathcal{C}$, x_i is outside of domain of input, return $\mathcal{B} = P_i$ (identified cheater) as output of $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$ to $\mathcal{A}$ and skip to simulation step of *Handling Abort*. Else run the following steps.

  – *Output of $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$ to $\mathcal{A}$:* Choose random sh_j for $j \in \mathcal{C}$ and compute its authentication (Step 2, 3 of $f_{\mathsf{ASh}}^{\lfloor n/2\rfloor}$ in Figure 6.5). The resulting values $S_j = \{\mathsf{sh}_j, \{k_{ij}\}_{i\in[n]}, \{\mathsf{tag}_{ji}\}_{i\in[n]}\}$ are given to $\mathcal{A}$ as the outputs of the corrupted parties from functionality $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$. Note that functionality $\mathcal{F}_{\mathsf{god}}$ computing f has not been invoked yet.

  – If $\mathcal{A}$ invokes $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$ with $(\mathtt{abort}, \mathcal{B})$, proceed to simulation step of *Handling Abort*.

  – *Round 3 in case of no abort:* Else, if $\mathcal{A}$ invokes $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$ with `continue`, then invoke $\mathcal{F}_{\mathsf{god}}$ with $\{x_j\}_{j\in\mathcal{C}}$ on behalf of $\mathcal{A}$ to obtain output y. The following steps are used to simulate Round 3:

    1. Interpolate a $\lfloor n/2\rfloor$-degree polynomial $A(x)$ with $A(j) = \mathsf{sh}_j$ for $j \in \mathcal{C}$ and $A(0) = y$.
    2. Set $\mathsf{sh}_i = A(i)$ for $i \in \mathcal{H}$. Using k_{ij} (consistent with output of $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$), compute $\mathsf{tag}_{ij} = \mathsf{Mac}_{k_{ij}}(\mathsf{sh}_i)$. Send $(\mathsf{sh}_i, \mathsf{tag}_{ij})$ $(j \neq i)$ on behalf of P_i in Round 3.

– *Handling Abort.* $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$ does the following:

  – *Round 3:* Let $\mathsf{m}_i^2 \leftarrow \mathcal{S}_{\pi_{\mathsf{GOD}}}(\mathsf{T}_1, f^{\mathcal{B}}; r_i)$ $(i \in \mathcal{H})$ correspond to Round 2 message of P_i obtained by invoking $\mathcal{S}_{\pi_{\mathsf{GOD}}}$ with randomness r_i, function to be computed $f^{\mathcal{B}}$ and transcript of Round 1 i.e $\mathsf{T}_1 = \{\mathsf{m}_j^1\}_{j\in\mathcal{C}}$. Send m_i^2 on behalf of P_i to $\mathcal{A}$. Receive m_j^2 as Round 2 message of π_{GOD} sent by P_j ($j \in \mathcal{C}$). When $\mathcal{S}_{\pi_{\mathsf{GOD}}}$ returns the extracted input $\{x'_j\}_{j\in\mathcal{C}}$ of the corrupt party to invoke its ideal functionality $\mathcal{F}_{\mathsf{god}}$, $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$ invokes $\mathcal{F}_{\mathsf{god}}$ with input $\{x'_j\}_{j\in\mathcal{C}\setminus\mathcal{B}}$ on behalf of corrupt P_j (not identified among set of cheaters) and default input on behalf of parties in $\mathcal{B}$. Then, forward the obtained output y' as response to $\mathcal{S}_{\pi_{\mathsf{GOD}}}$.

  – *Round 4:* Let $\mathsf{m}_i^3 \leftarrow \mathcal{S}_{\pi_{\mathsf{GOD}}}(\mathsf{T}_2, y'; r_i)$ $(i \in \mathcal{H})$ correspond to Round 3 message of P_i obtained by invoking $\mathcal{S}_{\pi_{\mathsf{GOD}}}$ with randomness r_i and transcript upto Round 2 i.e $\mathsf{T}_2 = \{\mathsf{m}_j^1, \mathsf{m}_j^2\}_{j\in\mathcal{C}}$ and output y' of its ideal functionality. Send m_i^3 on behalf of P_i to $\mathcal{A}$ in Round 4, completing the simulation.

Figure 6.15: Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$

Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$ wrt $(t_a, t_p) = (\lceil n/2\rceil - 1, \lfloor n/2\rfloor)$: Let $\mathcal{A}$ be a boundary-admissible malicious adversary with corruption parameters $(t_a, t_p) = (\lceil n/2\rceil - 1, \lfloor n/2\rfloor)$ in the $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$-hybrid model execution of $\pi_{\mathsf{GOD}}^{\mathsf{bou}}$. The simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$, running an ideal-world evaluation of the functionality

$\mathcal{F}_{\mathsf{god}}$ (refer Figure 2.4) computing f whose behaviour simulates the behaviour of $\mathcal{A}$ is described in Figure 6.15. There are 2 different scenarios based on whether $\mathcal{A}$ aborts the computation of $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$. Incase abort doesn't occur, it follows directly from the properties of privacy of authenticated sharing (Lemma 6.13) that the view in the ideal world is indistinguishable to the view of $\mathcal{A}$ in the hybrid-world ($\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$-hybrid model execution of $\pi_{\mathsf{GOD}}^{\mathsf{bou}}$). The additional step of Round 1 of π_{GOD} being executed in Round 1 of $\pi_{\mathsf{GOD}}^{\mathsf{bou}}$ is simulated identical to $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh}}$. In case of abort, it follows from privacy of authenticated sharing $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor}$ that output of $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$ can be simulated without invoking $\mathcal{F}_{\mathsf{god}}$. This indicates that $\mathcal{A}$ who can potentially participate with a different set of inputs in π_{GOD} (compared to π_{idua}) will have no information about the output based on its inputs in π_{idua}. Lastly, in this abort case, we note that the only difference between the ideal and hybrid-execution is that the messages of π_{GOD} are obtained via the simulator $\mathcal{S}_{\pi_{\mathsf{GOD}}}$ in the former. Indistinguishability thus follows from the security of subprotocol π_{GOD}.

6.6.3.3 Proof of Security of $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$ (Theorem 6.7)

We prove Theorem 6.7 by presenting two separate simulators $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh},1}$ and $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal},1}$ for the case of $(t_a, t_p) = (0, n-1)$ and $(t_a, t_p) = (1, \lfloor n/2 \rfloor)$ respectively. $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh},1}$ and $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal},1}$ invoke the simulator of the subprotocol π_{idua}, say $\mathcal{S}_{\pi_{\mathsf{idua}}}$ (running an ideal-world evaluation of functionality $\mathcal{F}_{\mathsf{idua}}$, refer Figure 2.5).

Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh},1}$ wrt $(t_a, t_p) = (0, n-1)$: Let $\mathcal{A}$ be a boundary-admissible passive adversary with parameters $(t_a, t_p) = (0, n-1)$ in the execution of $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$. The simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh},1}$, running an ideal-world evaluation of the functionality $\mathcal{F}_{\mathsf{god}}$ (refer Figure 2.4) computing f whose behaviour simulates the behaviour of $\mathcal{A}$ is described in Figure 6.16. Since $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$ proceeds in the same manner as $\pi_{\mathsf{GOD}}^{\mathsf{bou}}$ in this case of pure passive corruptions, the simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh},1}$ executes similar steps as simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh}}$ (Figure 6.14). The only difference is that instead of analysis in the $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$-hybrid model, the simulator $\mathcal{S}_{\pi_{\mathsf{idua}}}$ is invoked to simulate messages of honest parties in Round 1 and Round 2. Note that the simulation of Round 1 of π_{GOD} in $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh}}$ is skipped here and instead an additional instance of Round 1 of π_{idua} is simulated. It thus follows from security of π_{idua} and the arguments wrt $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh}}$ that the view of $\mathcal{A}$ in the ideal world is indistinguishable to the view of $\mathcal{A}$ in the execution of $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$.

Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh},1}$

Let $\mathcal{C}, \mathcal{H}$ be the set of indices of corrupt and honest parties respectively. The following steps are carried out by $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh},1}$:

– *Invoking $\mathcal{F}_{\mathsf{god}}$:* Receive $\{x_i\}_{i \in \mathcal{C}}$ corresponding to the parties controlled by semi-honest adversary $\mathcal{A}$. Invoke $\mathcal{F}_{\mathsf{god}}$ on behalf of $\mathcal{A}$ with $\{x_i\}_{i \in \mathcal{C}}$ to receive an output value y in return.

– *Interaction with* $\mathcal{S}_{\pi_{\mathsf{idua}}}$*:* Compute the authenticated secret-sharing of value y with threshold $\lfloor n/2 \rfloor$ as $(S_1 \dots S_n) = f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor}(y)$ (Figure 6.5).

Round 1: For $k = 1, 2$, let $\mathsf{m}_i^{1,k} \leftarrow \mathcal{S}_{\pi_{\mathsf{idua}}}(r_i^k)$ $(i \in \mathcal{H})$ correspond to the Round 1 message of P_i obtained by invoking $\mathcal{S}_{\pi_{\mathsf{idua}}}$ with P_i's randomness r_i^k. Send $\{\mathsf{m}_i^{1,k}\}_{k\in[2]}$ on behalf of P_i to $\mathcal{A}$ in Round 1. Receive $\{\mathsf{m}_j^{1,k}\}_{k\in[2]}$ sent by P_j $(j \in \mathcal{C})$.

When $\mathcal{S}_{\pi_{\mathsf{idua}}}$ invokes its ideal functionality $\mathcal{F}_{\mathsf{idua}}$ computing $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor} \diamond f$ with input x_i, send output $S_j = \big(\mathsf{sh}_j, \{k_{ij}\}_{i\in[n]}, \{\mathsf{tag}_{ji}\}_{i\in[n]}\big)$ corresponding to P_j $(j \in \mathcal{C})$ as response from $\mathcal{F}_{\mathsf{idua}}$.

Round 2: Let $\mathsf{m}_i^{2,1} \leftarrow \mathcal{S}_{\pi_{\mathsf{idua}}}(\mathsf{T}_1, y', f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor} \diamond f, \mathcal{P}; r_i^1)$ $(i \in \mathcal{H})$ correspond to Round 2 message of P_i obtained by invoking $\mathcal{S}_{\pi_{\mathsf{idua}}}$ with randomness r_i^1 and transcript of Round 1 (first instance $k = 1$) i.e $\mathsf{T}_1 = \{\mathsf{m}_j^{1,1}\}_{j\in\mathcal{C}}$, $y' = \{S_j\}_{j\in\mathcal{C}}$ i.e output of $\mathcal{F}_{\mathsf{idua}}$, $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor} \diamond f$ as the function to be computed and $\mathcal{P}$ as the set of parties involved in computation. Send $\mathsf{m}_i^{2,1}$ on behalf of P_i to $\mathcal{A}$. Receive $\mathsf{m}_j^{2,1}$ sent by P_j $(j \in \mathcal{C})$.

– *Round 3:* For each $i \in \mathcal{H}$, send $(\mathsf{sh}_i, \mathsf{tag}_{ij})$ $(j \neq i)$ on behalf of P_i.

Figure 6.16: Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{sh},1}$

Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal},1}$

Let $\mathcal{C} \subset [n]$ and $\mathcal{H} = [n] \setminus \mathcal{C}$ be the set of indices of the corrupt parties and the honest parties respectively. The following steps are carried out by $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal},1}$:

– *Round 1:* For $k = 1, 2$, let $\mathsf{m}_i^{1,k} \leftarrow \mathcal{S}_{\pi_{\mathsf{idua}}}(r_i^k)$ $(i \in \mathcal{H})$ correspond to the Round 1 message of P_i obtained by invoking $\mathcal{S}_{\pi_{\mathsf{idua}}}$ with randomness r_i^k. Send $\{\mathsf{m}_i^{1,k}\}_{k\in 2}$ on behalf of P_i to $\mathcal{A}$. Receive $\{\mathsf{m}_j^{1,k}\}_{k\in[2]}$ sent by P_j $(j \in \mathcal{C})$.

– *Round 2:* Send $\mathsf{m}_j^{1,1}$ to $\mathcal{S}_{\pi_{\mathsf{idua}}}$ on behalf of P_j. When $\mathcal{S}_{\pi_{\mathsf{idua}}}$ returns the extracted input $\{x_j\}_{j\in\mathcal{C}}$ of the corrupt party to invoke its ideal functionality $\mathcal{F}_{\mathsf{idua}}$ computing $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor} \diamond f$, $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal},1}$ does the following:

  - If there exists a $j \in \mathcal{C}$ such that $x_j = \perp$, send $y' = (\perp, P_j)$ to $\mathcal{A}$ as output response of $\mathcal{F}_{\mathsf{idua}}$.
  - Else choose random sh_j for $j \in \mathcal{C}$ and compute its authentication (Step 2, 3 of $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor}()$ of Figure 6.5). The resulting values $S_j = \{\mathsf{sh}_j, \{k_{ij}\}_{i\in[n]}, \{\mathsf{tag}_{ji}\}_{i\in[n]}\}$ for each $j \in \mathcal{C}$ are given to $\mathcal{A}$ as the output from functionality $\mathcal{F}_{\mathsf{idua}}$.

  Let $\mathsf{m}_i^{2,1} \leftarrow \mathcal{S}_{\pi_{\mathsf{idua}}}(\mathsf{T}_1, y', f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor} \diamond f, \mathcal{P}; r_i^1)$ $(i \in \mathcal{H})$ correspond to the Round 2 message of P_i obtained by invoking $\mathcal{S}_{\pi_{\mathsf{idua}}}$ with P_i's randomness r_i^1, transcript of Round 1 (first instance $k = 1$) i.e $\mathsf{T}_1 = \{\mathsf{m}_j^{1,1}\}_{j\in\mathcal{C}}$, output $y' = \{S_j\}_{j\in\mathcal{C}}$ of $\mathcal{F}_{\mathsf{idua}}$, $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor} \diamond f$ as the function to be

computed and $\mathcal{P}$ as the set of parties involved in computation. Send $\mathsf{m}_i^{2,1}$ on behalf of P_i to $\mathcal{A}$. Receive $\mathsf{m}_j^{2,1}$ sent by P_j ($j \in \mathcal{C}$) and send it to $\mathcal{S}_{\pi_{\mathsf{idua}}}$ on behalf of P_j.

– *Round 3:* We now have 2 cases -

- If $\mathcal{S}_{\pi_{\mathsf{idua}}}$ invokes $\mathcal{F}_{\mathsf{idua}}$ with $(\mathtt{abort}, \mathcal{B})$, do the following: Send $\mathsf{m}_j^{1,2}$ to $\mathcal{S}_{\pi_{\mathsf{idua}}}$ on behalf of $\mathcal{A}$ ($P_j \in \mathcal{E}$) corresponding to second instance of π_{idua}. Suppose $\{x_j\}_{j\in\mathcal{C}}$ is the extracted input returned by $\mathcal{S}_{\pi_{\mathsf{idua}}}$, then invoke $\mathcal{F}_{\mathsf{god}}$ with $\{x_j\}_{j\in\mathcal{C}}$ and substituting default input of party in $\mathcal{B}$; on behalf of $\mathcal{A}$ to obtain output y. Let $\mathsf{m}_i^{2,2} \leftarrow \mathcal{S}_{\pi_{\mathsf{idua}}}(\mathsf{T}_1, y, f^{\mathcal{B}}, \mathcal{P} \setminus \mathcal{B}; r_i^2)$ ($i \in \mathcal{H}$) correspond to the Round 2 message of π_{idua} obtained by invoking $\mathcal{S}_{\pi_{\mathsf{idua}}}$ with P_i's randomness r_i^2, transcript of Round 1 (second instance $k = 2$) i.e $\mathsf{T}_1 = \{\mathsf{m}_j^{1,2}\}_{j\in\mathcal{C}}$, output y, $f^{\mathcal{B}}$ as function to be computed (same as f except with default inputs hardcoded for the parties in $\mathcal{B}$) and $\mathcal{P} \setminus \mathcal{B}$ as the set of parties involved in computation. Send $\mathsf{m}_i^{2,2}$ on behalf of P_i to $\mathcal{A}$. Receive $\mathsf{m}_j^{2,2}$ sent by P_j ($j \in \mathcal{C}$) and send it to $\mathcal{S}_{\pi_{\mathsf{idua}}}$ on behalf of P_j.
- If $\mathcal{S}_{\pi_{\mathsf{idua}}}$ invokes $\mathcal{F}_{\mathsf{idua}}$ with `continue`, run the same steps as Round 3 simulation incase of no abort of $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$ (Figure 6.15).

Figure 6.17: Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal},1}$

Simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal},1}$ wrt $(t_a, t_p) = (1, \lfloor n/2 \rfloor)$: Let $\mathcal{A}$ be a malicious adversary controlling atmost 1 party actively and upto $\lfloor n/2 \rfloor$ parties passively in an execution of $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$. The simulator $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal},1}$, running an ideal-world evaluation of the functionality $\mathcal{F}_{\mathsf{god}}$ (refer Figure 2.4) computing f whose behaviour simulates the behaviour of $\mathcal{A}$ is described in Figure 6.17. There are 2 different scenarios based on whether $\mathcal{A}$ aborts the computation in first instance of π_{idua}. Incase abort doesn't occur, simulation proceeds similar to $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$ (Figure 6.15) except that instead of analysis in $\mathcal{F}_{\mathsf{idua}}^{\mathsf{ASh}}$- hybrid model, the simulator $\mathcal{S}_{\pi_{\mathsf{idua}}}$ is invoked for simulation in Round 1 and Round 2. Another difference is that an additional instance of Round 1 of π_{idua} is simulated. Thus, in case of no abort, it follows from the security argument wrt $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$ and the security of π_{idua} that the view of $\mathcal{A}$ in the ideal world is indistinguishable to the view of $\mathcal{A}$ in the execution of $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$.

Consider case of abort which returns the identity of cheater, say singleton set $\mathcal{B}$, (π_{idua} realizes $\mathcal{F}_{\mathsf{idua}}$). Another execution of π_{idua} is used to compute the function $f^{\mathcal{B}}$ (same as f except that it hardcodes default input of the actively corrupt party identified). Accordingly, Round 3 of $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$ is simulated by invoking $\mathcal{S}_{\pi_{\mathsf{idua}}}$ to obtain Round 2 of π_{idua} wrt function $f^{\mathcal{B}}$ among parties in $\mathcal{P} \setminus \mathcal{B}$ (wrt second instance of Round 1 of π_{idua} run in Round 1 of $\pi_{\mathsf{GOD}}^{\mathsf{bou},1}$). Note that the output of first instance of π_{idua} was simulated perfectly by $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal},1}$ without invoking $\mathcal{F}_{\mathsf{god}}$ (relying on privacy of $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor}()$, similar to argument in $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$), implying that $\mathcal{A}$ had no information about

the output of f at the end of Round 2. It is now easy to check that the only difference that remains between the ideal and the real execution is that the messages of honest parties are obtained via $\mathcal{S}_{\pi_{\mathsf{idua}}}$ in the ideal world. We can thus conclude that the view of $\mathcal{A}$ in the ideal world is indistinguishable to the view of $\mathcal{A}$ in the execution of $\pi^{\mathsf{bou},1}_{\mathsf{GOD}}$ based on the security of π_{idua} and the security arguments presented wrt $\mathcal{S}^{\mathsf{mal}}_{\mathsf{GOD}}$. This completes the proof.

6.6.3.4 Proof of Security of $\pi^{\mathsf{bou}}_{\mathsf{fn}}$ (Theorem 6.8)

We prove Theorem 6.8 by presenting two separate simulators $\mathcal{S}^{\mathsf{sh}}_{\mathsf{fn}}$ and $\mathcal{S}^{\mathsf{mal}}_{\mathsf{fn}}$ for the case of corruption scenarios $(t_a, t_p) = (0, n-1)$ and $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ respectively. The protocol $\pi^{\mathsf{bou}}_{\mathsf{fn}}$ is analyzed in a $\mathcal{F}^{\mathsf{ASh}}_{\mathsf{ua}}$-hybrid model where the parties have access to a trusted party computing $\mathcal{F}^{\mathsf{ASh}}_{\mathsf{ua}}$ (Figure 6.13).

Simulator $\mathcal{S}^{\mathsf{sh}}_{\mathsf{fn}}$ wrt $(t_a, t_p) = (0, n-1)$: Let $\mathcal{A}$ be the boundary-admissible passive adversary controlling upto $(n-1)$ parties in the $\mathcal{F}^{\mathsf{ASh}}_{\mathsf{ua}}$-hybrid model execution of $\pi^{\mathsf{bou}}_{\mathsf{fn}}$. The simulator $\mathcal{S}^{\mathsf{sh}}_{\mathsf{fn}}$, running an ideal-world evaluation of the functionality $\mathcal{F}_{\mathsf{god}}$ (refer Figure 2.4) computing f whose behaviour simulates the behaviour of $\mathcal{A}$ is described in Figure 6.18. It directly follows from the security arguments presented wrt $\mathcal{S}^{\mathsf{sh}}_{\mathsf{GOD}}$ (Figure 6.14), that the view of $\mathcal{A}$ in the ideal world is identical to the view of $\mathcal{A}$ in the $\mathcal{F}^{\mathsf{ASh}}_{\mathsf{ua}}$-hybrid model execution of $\pi^{\mathsf{bou}}_{\mathsf{fn}}$.

Simulator $\mathcal{S}^{\mathsf{sh}}_{\mathsf{fn}}$

Let $\mathcal{C} \subset [n]$, $\mathcal{H}$ denote the set of indices of corrupt and honest parties respectively. The following steps are carried out by $\mathcal{S}^{\mathsf{sh}}_{\mathsf{fn}}$:

- *Invoking $\mathcal{F}_{\mathsf{god}}$:* Receive $\{x_j\}_{j \in \mathcal{C}}$ corresponding to the parties controlled by passive adversary $\mathcal{A}$. Invoke $\mathcal{F}_{\mathsf{god}}$ on behalf of $\mathcal{A}$ with $\{x_j\}_{j \in \mathcal{C}}$ to receive an output value y in return.
- *Output of $\mathcal{F}^{\mathsf{ASh}}_{\mathsf{ua}}$ to $\mathcal{A}$:* Compute the authenticated secret-sharing of value y with threshold $t = \lfloor n/2 \rfloor$ (Figure 6.5) as $(S_1 \ldots S_n) = f^{\lfloor n/2 \rfloor}_{\mathsf{ASh}}(y)$ and send $S_j = \left(\mathsf{sh}_j, \{k_{ij}\}_{i \in [n]}, \{\mathsf{tag}_{ji}\}_{i \in [n]}\right)$ as output of $\mathcal{F}^{\mathsf{ASh}}_{\mathsf{ua}}$ to P_j $(j \in \mathcal{C})$.
- *Round 3:* Broadcast $(\mathsf{sh}_i, \mathsf{tag}_{ij})$ on behalf of P_i for each $i \in \mathcal{H}, j \neq i$.

Figure 6.18: Simulator $\mathcal{S}^{\mathsf{sh}}_{\mathsf{fn}}$

Simulator $\mathcal{S}^{\mathsf{mal}}_{\mathsf{fn}}$ wrt $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$: Let $\mathcal{A}$ be a malicious adversary with corruption parameters $(t_a, t_p) = (\lceil n/2 \rceil - 1, \lfloor n/2 \rfloor)$ parties in the $\mathcal{F}^{\mathsf{ASh}}_{\mathsf{ua}}$-hybrid model execution of $\pi^{\mathsf{bou}}_{\mathsf{fn}}$. The simulator $\mathcal{S}^{\mathsf{mal}}_{\mathsf{fn}}$, running an ideal-world evaluation of the functionality $\mathcal{F}_{\mathsf{fair}}$ (Figure 2.3) computing f whose behaviour simulates the behaviour of $\mathcal{A}$ is described in Figure 6.19. There are 2 different scenarios based on whether $\mathcal{A}$ aborts the computation of $\mathcal{F}^{\mathsf{ASh}}_{\mathsf{ua}}$. Incase of abort,

it follows from privacy of sharing function $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor}$ that the view of $\mathcal{A}$ in the ideal world is indistinguishable to the $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$-hybrid model execution of $\pi_{\mathsf{fn}}^{\mathsf{bou}}$. In case of no abort, the simulation proceeds similar to $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$ (Figure 6.15, no abort case). We can thus conclude based on the security arguments of $\mathcal{S}_{\mathsf{GOD}}^{\mathsf{mal}}$ and the properties of authenticated secret-sharing that the view of $\mathcal{A}$ in the ideal world is indistinguishable to the $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$- hybrid model execution of $\pi_{\mathsf{fn}}^{\mathsf{bou}}$. This completes the proof.

Simulator $\mathcal{S}_{\mathsf{fn}}^{\mathsf{mal}}$

Let $\mathcal{C} \subset [n]$ and $\mathcal{H} = [n] \setminus \mathcal{C}$ be the set of indices corrupt and honest parties respectively. The following steps are carried out by $\mathcal{S}_{\mathsf{fn}}^{\mathsf{mal}}$:

– *Interaction of $\mathcal{A}$ with $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$:* Receive $\{x_j\}_{j \in \mathcal{C}}$ sent by malicious $\mathcal{A}$ to $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$ in this $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$-hybrid execution model. If for any $j \in \mathcal{C}$, x_j is outside of domain of input, send $\perp$ as output of $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$ to $\mathcal{A}$ and send $\perp$ as input to $\mathcal{F}_{\mathsf{fair}}$ on behalf of $\mathcal{A}$. Else run the following steps.

– *Output of $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$ to $\mathcal{A}$:* Choose random sh_j for $j \in \mathcal{C}$ and compute its authentication (Step 2, 3 of $f_{\mathsf{ASh}}^{\lfloor n/2 \rfloor}$ in Figure 6.5). The resulting values $S_j = \{\mathsf{sh}_j, \{k_{ij}\}_{i \in [n]}, \{\mathsf{tag}_{ji}\}_{j \in [n]}\}$ are given to $\mathcal{A}$ as the outputs of the corrupted parties from functionality $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$. Note that functionality $\mathcal{F}_{\mathsf{fair}}$ computing f has not been invoked yet.

– *Invoking $\mathcal{F}_{\mathsf{fair}}$:* We have 2 cases based on whether $\mathcal{A}$ invokes $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$ with `abort` or `continue`.

- `abort`: Send $\perp$ as input to $\mathcal{F}_{\mathsf{fair}}$ on behalf of $\mathcal{A}$; thereby completing the simulation.
- `continue`: Invoke $\mathcal{F}_{\mathsf{fair}}$ with $\{x_j\}_{j \in \mathcal{C}}$ on behalf of $\mathcal{A}$ to obtain y.

– *Round 3:* The following steps are used to simulate Round 3 -

- Interpolate a $\lfloor n/2 \rfloor$-degree polynomial $A(x)$ with $A(j) = \mathsf{sh}_j$ for $j \in \mathcal{C}$ and $A(0) = y$.
- Set $\mathsf{sh}_i = A(i)$ for $i \in \mathcal{H}$. Using k_{ij} (consistent with the output of $\mathcal{F}_{\mathsf{ua}}^{\mathsf{ASh}}$ sent to $\mathcal{A}$), compute $\mathsf{tag}_{ij} = \mathsf{Mac}_{k_{ij}}(\mathsf{sh}_i)$.
- Send $(\mathsf{sh}_i, \mathsf{tag}_{ij})$ $(i \in \mathcal{H})$ on behalf of P_i $(j \neq i)$.

Figure 6.19: Simulator $\mathcal{S}_{\mathsf{fn}}^{\mathsf{mal}}$

Part III

Secure Computation in Hybrid Networks

Chapter 7

On the Power of Hybrid Networks in Multi-Party Computation

In this final chapter, we depart from the computational and synchronous setting of the previous chapters of the thesis and explore the information-theoretic setting where the adversary is computationally unbounded and explore asynchrony in the network as well. Specifically, we investigate perfectly-secure VSS and MPC protocols in hybrid networks that combine the best features of protocols in the synchronous and asynchronous networks.

7.1 Introduction

In the literature, VSS and MPC have been explored in two prominent network settings: *synchronous* and *asynchronous* networks. Recall that in the synchronous setting (also known as bounded-synchronous [83, 175, 100, 13]), it is assumed that the delay of messages in the channels of the network is bounded by a *known constant*. This allows protocols to proceed in rounds, with the strong delivery guarantee that every message sent in any given round is delivered to all the recipients in the same round. In contrast, in the asynchronous setting, the channels in the network may have arbitrary delays and may deliver messages in any arbitrary order, with the only restriction that every sent message must eventually be delivered. In order to model the worst case, the adversary is allowed to control the scheduling of messages in the network.

The synchronous network is well-behaved and convenient, but unrealistic and inapplicable in many practical environments. Whereas, the asynchronous network can aptly model real-life networks like Internet, is difficult to deal with and less convenient. When the channel delays are short, the protocols in asynchronous networks may be faster than synchronous protocols which have to allow each round to be long enough, such that all messages can get through, even

in the very worst case. As a result, softwares based on practical implementation of synchronous protocols must choose actual waiting time correctly with respect to the network used. This makes the software dependant on the underlying network which is clearly undesirable, as it creates an extra source of errors, insecurity, or both [75]. Therefore, many practical frameworks follow an asynchronous communication model. For instance, VIFF (Virtual Ideal Functionality Framework) [75] is basically asynchronous and operates on the principle that parties proceed whenever possible and allows automatic parallel scheduling of the operations, i.e. the programmer does not have to specify any explicit timing of operations. Sharemind [39], a framework for fast privacy-preserving computations is also based on asynchronous communication model that allows to omit the central synchronisation service and thus reduce network delays that have significant impact on the overall efficiency. Finally, implementations with network layer based on UDP networking must account for a malicious network scheduler that can drop messages or change their order, which is appropriately modelled by the asynchronous network [38]. These clearly substantiate that asynchronous network is more realistic and in many cases results in faster protocols than synchronous network.

On the downside, asynchronous protocols suffer from low fault-tolerance, high communication complexity and relatively weaker guarantees compared to their synchronous counterparts. The asynchronous VSS suffers from *dealer-dependent termination* where termination of the sharing phase is guaranteed only when the dealer is honest. Whereas, synchronous VSS guarantees termination independent of the status of the dealer. Similarly, asynchronous MPC suffers from *input deprivation* that refers to a property where inputs of upto t honest parties may be excluded from computation. Whereas, synchronous MPC offers *input provision* which refers to inclusion of the inputs of all the honest parties for computation, apart from giving the other properties of asynchronous MPC. All the above issues are supposedly caused by the following inherent and trademark difficulty in the asynchronous model. In an asynchronous network, an honest party whose message is delayed in the network cannot be told apart from a corrupted party who did not send a message at all. So an honest party in an asynchronous protocol, unlike in a synchronous protocol, cannot wait for the messages from all the parties, as it would potentially risk him to wait infinitely. To avoid the risk, an honest party's computation in an asynchronous protocol should be carried on with the receipt of $(n - t)$ parties at any given step. Unfortunately, this may risk ignoring the values of up to t potentially honest parties at any given step. In what follows, we highlight the well-known gaps in the feasibility results of the synchronous and asynchronous VSS and MPC that corroborate with the above inherent difficulty faced in asynchronous protocols.

7.1.1 Related Work

It is known that the feasibility of asynchronous protocols holds under stricter conditions than that of synchronous protocols. Perfect (error-free) asynchronous MPC (and VSS) requires $t < n/4$ [31], whereas perfect synchronous MPC (and VSS) is feasible with $t < n/3$ [30, 79]. Statistical and computational asynchronous MPC (and VSS) protocols require $t < n/3$ [51, 32, 48], whereas their synchronous counterparts are feasible with $t < n/2$ [83, 177, 175]. Though our work mainly concerns the theoretical feasibility gap between the two classes, the following results on communication efficiency further substantiate that the state-of-the-art protocols in asynchronous world lag far behind. The best known perfect MPC protocol in the synchronous and asynchronous network achieves communication complexity $\mathcal{O}(|C|n|\mathbb{F}|)$ [21] and $\mathcal{O}(|C|n^2|\mathbb{F}|)$ [174] bits respectively. Here $|C|$ denotes the number of multiplication gates in the arithmetic circuit C representing the function to be computed and $\mathbb{F}$ denotes the underlying field. The gap is noticeably wider in the statistical case. For a statistical security parameter μ, it is $\mathcal{O}(|C|n\mu)$ bits [33] versus $\mathcal{O}(|C|n^5\mu)$ bits [173]. The situation is slightly promising in the cryptographic setting. For a security parameter denoted as κ, the best protocols in both the worlds achieve $\mathcal{O}(|C|n\kappa)$ bits of communication complexity [117, 61]. However, while the synchronous protocol of [117] relies on homomorphic encryption, the protocol of [61] uses *somewhat homomorphic encryption (SHE)*. A summary of the above results are given below.

Security	Network	Resilience	Communication Complexity
Perfect	Synchronous	$t < n/3$ [30]	$\mathcal{O}(\|C\|n\|\mathbb{F}\|)$ [21]
	Asynchronous	$t < n/4$ [31]	$\mathcal{O}(\|C\|n^2\|\mathbb{F}\|)$ [174]
Statistical	Synchronous	$t < n/2$ [177]	$\mathcal{O}(\|C\|n\mu)$ [33]
	Asynchronous	$t < n/3$ [32]	$\mathcal{O}(\|C\|n^5\mu)$ [173]

Bridging the Gaps While no effort has been made thus far to close the gaps in fault-tolerance of VSS protocols in any setting, MPC literature has seen a few attempts. The first attempt to bridge the feasibility gap between synchronous and asynchronous MPC was made by [23]. Their cryptographic MPC protocol provides input provision and works with $t < n/2$ which is the same bound necessary and sufficient for synchronous cryptographic MPC. This is achieved at the expense of one initial synchronous round that allows access to broadcast oracle in the *hybrid* network setting. A network that is asynchronous in nature and yet supports a few synchronous rounds at the onset of a protocol execution is denoted as hybrid network. In an alternative approach, [75] introduces a *synchronisation* point (the network is asynchronous before and after

the point) and presents a perfect asynchronous MPC protocol with $t < n/3$. MPC protocol in the synchronisation point model inherently do not guarantee termination and outputs to all honest parties. In yet another approach, [14] uses non-equivocation technique to achieve asynchronous MPC with $t < n/2$ that matches the bound needed for cryptographic synchronous MPC. Non-equivocation is a message authentication mechanism to restrict a corrupted sender from making conflicting statements to different (honest) parties and requires cryptography for realization. [63, 62] shows how to bridge communication-efficiency gap for perfect MPC with the help of a single synchronous round. There have also been attempts to improve feasibility bounds of broadcast [12, 84, 153] which is a special case of MPC and a fundamental problem in distributed computing, by assuming slightly more powerful communication models. [85, 68] achieve improved fault resilience of unconditional broadcast by assuming existence of partial broadcast channel among subset of parties.

7.1.2 Our Results

Taking cognizance of the fact that asynchronous network is more realistic and may result in faster protocols on one hand and on the other, synchrony of a network has positive impact on several aspects of distributed protocols, we set our focus on the power of hybrid networks that combines best of both the worlds. In hybrid networks, we wish to investigate feasibility of protocols giving guarantees attainable in synchronous as well as asynchronous networks. Denoting synchronous/asynchronous VSS (SVSS/AVSS) and synchronous/asynchronous MPC (SMPC/AMPC) to refer to the properties of the protocols that can be achieved in the respective networks, we present our findings below. The notations should not be confused with the underlying network. All our results are based in the hybrid network model. For asynchronous protocols, we wish to bridge the fault-tolerance gap between synchronous and asynchronous protocols with minimum synchrony assumption needed, leveraging the initial synchronous rounds. For synchronous protocols, we explore if the known lower bounds on round complexity can be circumvented, leveraging the asynchronous phase available in the hybrid network. While the details of our results appear in Section 1.4.4, we summarize them below.

For the asynchronous protocols in hybrid networks, we hope to leverage the initial synchronous rounds to bridge the gap in the fault-tolerance with the synchronous protocols under minimal synchrony assumption. We ask the following fundamental question of both theoretical and practical importance: *What is the minimum number of initial synchronous rounds necessary and sufficient in a hybrid network to construct asynchronous perfectly-secure VSS and MPC protocols with the fault-tolerance of synchronous protocols?* On the positive note, we show that the answer is *one* for VSS which is clearly optimal. Notably *no broadcast* oracle is invoked

in the synchronous round of our proposed VSS protocol. On the negative side, we prove that one synchronous round is *not* enough for MPC, putting MPC on a higher pedestal than VSS in terms of difficulty.

For synchronous protocols in hybrid networks, we hope to save on the synchronous rounds leveraging conveniently the available asynchronous phase. We settle the question for VSS in the *negative* showing that three rounds that are known to be necessary (and sufficient) for VSS in synchronous networks, are also required in hybrid networks. VSS being a special case of MPC, the lower bound holds true for MPC. We match the lower bound with a 3-round protocol. Notably, synchronous MPC with cryptographic security is known to be achievable in hybrid networks with one synchronous round.

We summarize the feasibility results of SVSS/AVSS and SMPC/AMPC in hybrid networks in terms of initial synchronous rounds needed in the table below. Finding a tight upper bound for AMPC with two rounds remains an interesting open question.

Table 7.1: Feasibility for SVSS/AVSS and SMPC/AMPC with $t < n/3$ in Hybrid networks

Security		Asynchronous	Synchronous
VSS	Necessary	One [Trivial]	Three **(Our Work)** [167]
	Sufficiency	One **(Our Work)** [167]	Three [101]
MPC	Necessary	Two **(Our Work)** [167]	Three **(Our Work)** [167]
	Sufficiency	Three **(Our Work)** [167]	Three **(Our Work)** [167]

Lastly, our results reinforce several general beliefs in the context of hybrid networks: **(a)** AMPC is harder to achieve than AVSS, **(b)** SVSS is harder to achieve than AVSS with the same resilience, **(c)** perfectly-secure SMPC is harder to achieve than cryptographic SMPC.

Roadmap. In Section 7.2, we describe the network and adversarial model, introduce some notation, recall the relevant definitions and discuss the primitives needed for our protocols. In Section 7.3, we present a novel primitive termed as asynchronous weak polynomial sharing (AWPS) protocol which will serve as a building block for our proposed AVSS protocol. In Section 7.4, we present our AVSS protocol. The impossibility results on AMPC and SVSS appear in Section 7.5 and 7.6 respectively. Section 7.7 presents our SMPC over hybrid network with three synchronous rounds.

7.2 Preliminaries

7.2.1 Model

We consider a set of $n \geq 3t + 1$ PPT parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, connected by pairwise secure and authentic channels. We assume that there exists a static computationally unbounded

adversary $\mathcal{A}$, who can actively corrupt at most t out of the n parties and make them behave in any arbitrary manner during the execution of a protocol. We further allow the adversary to be rushing, i.e. in every round (or step) it can wait to hear the messages of the honest parties before sending his own messages.

We consider a hybrid network that is asynchronous in nature and yet supports a few initial synchronous rounds. During the onset of a protocol execution, the network behaves like a synchronous network with a global clock, and the protocol proceed in rounds with strong delivery guarantee that messages sent by any party in any given round are delivered to all recipients in the same round. The channels have fixed delays. After the synchronous rounds, the underlying network turns to a complete asynchronous network that deliver messages in an arbitrary order and impose arbitrary delays on them. Specifically, the communication channels between the parties have arbitrary, yet finite delay (i.e. the messages are guaranteed to reach their destinations eventually). Moreover the order in which the messages reach their destinations may be different from the order in which they were sent. To model the worst case scenario, $\mathcal{A}$ is given the power to schedule the delivery of every message in the network. However, $\mathcal{A}$ can only schedule the messages communicated between the honest parties, without having any access to the "content" of these messages. More details on asynchronous model can be found in [49].

Notation. he computation in our protocols is performed over a finite field $\mathbb{F}$ such that $|\mathbb{F}| > n$. Every field element can be represented by $\log |\mathbb{F}|$ bits. A set of values $\mathsf{Set} = \{(i, s_i)\}$ of size at least $t+1$ is said to be t-consistent if there exists a polynomial over $\mathbb{F}$, say $f(x)$, of degree at most t, such that $f(i) = s_i$. We define an algorithm $\mathsf{constantTerm}$ that takes a set $\mathsf{Set} = \{(i, s_i)\}$ with at least $t + 1$ values and returns $f(0)$ if Set is t-consistent and $f(x)$ is the polynomial passing through the points, $\perp$ otherwise. When there are exactly $t + 1$ points, $\mathsf{constantTerm}$ always returns $f(0)$.

7.2.2 Definitions

Synchronous/asynchronous VSS (SVSS/AVSS) and synchronous/asynchronous MPC (SMPC / AMPC) refer to the properties of the primitives attainable in synchronous and asynchronous networks respectively as defined formally below. This should not be confused with the underlying network. All our results are based in the hybrid network model.

We next recall the definition of AVSS from [31, 49].

Definition 7.1 (Asynchronous Verifiable Secret Sharing (AVSS) [31, 49]) *A pair of protocols* $(\mathsf{Sh}, \mathsf{Rec})$ *for* n *parties* $\mathcal{P}$*, where a dealer* $\mathsf{D} \in \mathcal{P}$ *holds a private input* $s \in \mathbb{F}$ *for* Sh *is an AVSS scheme tolerating* $\mathcal{A}$ *if the following requirements hold for every possible* $\mathcal{A}$ *and for all*

possible inputs of D*:*

– **Termination***:*

1. *If* D *is honest and all the honest parties participate in the protocol* Sh*, then each honest party eventually terminates the protocol* Sh*.*
2. *If some honest party terminates* Sh*, then irrespective of the behavior of* D*, each honest party eventually terminates* Sh*.*
3. *If all the honest parties invoke* Rec*, then each honest party eventually terminates* Rec*.*

– **Correctness***: If some honest party terminates* Sh*, then there exists a fixed value* $\bar{s} \in \mathbb{F}$*, such that the following requirements hold. We refer to the above as '*D *has committed/shared* $\bar{s}$ *during* Sh*'.:*

1. *If* D *is honest then* $\bar{s} = s$ *and each honest party upon completing the protocol* Rec*, outputs* s*.*
2. *Even if* D *is corrupted, each honest party upon completing* Rec *outputs* $\bar{s}$*, irrespective of the behavior of the corrupted parties. This property is also known as strong commitment.*

– **Privacy***: If* D *is honest then the adversary's view during* Sh *reveals no information about* s *in the information-theoretic sense; i.e. the adversary's view is identically distributed for all possible* s*.*

The termination guarantee of AVSS is dealer-dependent. The sharing phase protocol needs to terminate when the dealer is honest. In contrast, the sharing phase of SVSS protocols must terminate irrespective of any adversarial behavior. We now formally define synchronous VSS (SVSS).

Definition 7.2 (Synchronous Verifiable Secret Sharing (SVSS) [59]**)** *A pair of protocols* $(\mathsf{Sh}, \mathsf{Rec})$ *for* n *parties* $\mathcal{P}$*, where a dealer* $\mathsf{D} \in \mathcal{P}$ *holds a private input* $s \in \mathbb{F}$ *for* Sh *is a SVSS scheme tolerating* $\mathcal{A}$ *if the following requirements hold for every possible* $\mathcal{A}$ *and for all possible inputs of* D*:*

– **Termination***: If the honest parties participate in* Sh *and* Rec*, the respective protocols must terminate.*

– **Correctness**: *There exists a fixed value $\bar{s} \in \mathbb{F}$, such that the following requirements hold.*

1. *If* D *is honest then* $\bar{s} = s$ *and each honest party upon completing the protocol* Rec*, outputs* s*.*
2. *Even if* D *is corrupted, each honest party upon completing* Rec *outputs* $\bar{s}$*, irrespective of the behavior of the corrupted parties. This property is also known as strong commitment.*

We refer to the above as 'D *has committed/shared* $\bar{s}$ *during* Sh*':*

– **Privacy**: *If* D *is honest then the adversary's view during* Sh *reveals no information on* s*. More formally, the adversary's view is identically distributed for all possible values of* s*.*

Next, we recall the definition of t-sharing and $2d$-sharing.

Definition 7.3 (t-sharing and $2d$-sharing) *A value $s \in \mathbb{F}$ is said to be t-shared among $\mathcal{P}$ if there exists a polynomial over $\mathbb{F}$, say $f(x)$, of degree at most t, such that $f(0) = s$ and every (honest) party P_i holds a share s_i of s, where $s_i = f(i)$. A value $s \in \mathbb{F}$ is said to be $2d$-shared among $\mathcal{P}$ if:*

– *s is t-shared among $\mathcal{P}$ with shares $(s_1, \ldots, s_n)$ and*

– *each share s_i is also t-shared among $\mathcal{P}$.*

We use $[s]$ and $\langle s \rangle$ to denote that s is t-shared and $2d$-shared respectively.

We now proceed to present the definition of AMPC and SMPC.

Definition 7.4 (Asynchronous Multi-Party Computation (AMPC) [32]) *Let $F : \mathbb{F}^n \to \mathbb{F}^n$ be a publicly known function and let party P_i have a private input $x_i \in \mathbb{F}$. Any AMPC consists of three stages. In the first stage, each party P_i commits its input. Even if P_i is faulty, if it completed this step, then it is committed to some value (not necessarily x_i). Let x'_i be the value committed by P_i. If P_i is honest then $x'_i = x_i$. Then the parties agree on a common subset C of at least $n - t$ committed inputs. In the last stage the parties compute $F(y_1, \ldots, y_n)$, where $y_i = x'_i$ if $P_i \in C$, otherwise $y_i = 0$.*

An asynchronous protocol π among the n parties for computing the function F is called an AMPC protocol if it satisfies the following conditions for every possible $\mathcal{A}$ and inputs of the honest parties:

– **Termination**: *If all the honest parties participate in the protocol, then every honest party eventually terminates π.*

– **Correctness**: *Every honest party outputs $F(y_1, \ldots, y_n)$ after completing π, irrespective of the behavior of the corrupted parties.*

– **Privacy**: *The adversary obtains no additional information (in the information-theoretic sense) about the inputs of the honest parties during π, other than what is inferred from the input and the output of the corrupted parties.*

While AMPC suffers from *input deprivation* that refers to a property where inputs of upto t honest parties may be excluded from computation, SMPC provides *input provision* and includes inputs of all the honest parties for computation. Moreover, the termination is implicit for perfectly-secure SMPC and is not defined separately.

Definition 7.5 (Synchronous Multi-Party Computation (SMPC)) *Let $F : \mathbb{F}^n \rightarrow \mathbb{F}^n$ be a publicly known function and let party P_i have a private input $x_i \in \mathbb{F}$. A protocol π among the n parties securely computes $y = F(x_1, \ldots, x_n)$, if it satisfies the following for every possible $\mathcal{A}$, on all possible inputs:*

– **Correctness**: *All honest parties obtain y at the end of π.*

– **Privacy**: *$\mathcal{A}$ obtains no additional information (in the information-theoretic sense) about the inputs of the honest parties during π, other than what is inferred from the input and the output of the corrupted parties.*

7.2.3 Primitives

7.2.3.1 Asynchronous Broadcast

In our protocols, we use the asynchronous broadcast primitive, which was introduced and elegantly implemented by Bracha [43]; the primitive allows a special party $\mathsf{S} \in \mathcal{P}$, called sender, to send a message identically to all the parties. More formally:

Definition 7.6 (Asynchronous Broadcast [51]) *Let Π be an asynchronous protocol for the n parties initiated by a special party $\mathsf{S} \in \mathcal{P}$, having input m (the message to be broadcast). We say that Π is an asynchronous broadcast protocol if the following hold, for every possible $\mathcal{A}$:*

– **Termination**:

1. *If* S *is honest and all the honest parties participate in the protocol, then each honest party eventually terminates the protocol.*
2. *Irrespective of the behavior of* S*, if any honest party terminates the protocol then each honest party eventually terminates the protocol.*

– **Correctness**: *If the honest parties terminate the protocol then they do so with a common output $m^\star$. Furthermore, if the sender is honest then $m^\star = m$.*

Bracha presented a protocol called a-cast, for realizing the asynchronous broadcast primitive; the protocol can tolerate upto $t < n/3$ corruptions. For the sake of completeness, we recall the Bracha's a-cast protocol from [49] and present it in Figure 7.1.

Protocol a-cast()

– **Input of S:** A message m.

1. **Code for the sender S:** — only S executes this code:
   (a) Send the message $(\texttt{MSG}, m)$ to all the parties.
2. **Code for the party P_i:** — every party in $\mathcal{P}$ (including S) executes this code:
   (a) Upon receiving a message $(\texttt{MSG}, m)$ from S, send the message $(\texttt{ECHO}, m)$ to all the parties.
   (b) Upon receiving $n - t$ messages $(\texttt{ECHO}, m^\star)$ that agree on the value of $m^\star$, send the message $(\texttt{READY}, m^\star)$ to all the parties.
   (c) Upon receiving $t + 1$ messages $(\texttt{READY}, m^\star)$ that agree on the value of $m^\star$, send the message $(\texttt{READY}, m^\star)$ to all the parties.
   (d) Upon receiving $n - t$ messages $(\texttt{READY}, m^\star)$ that agree on the value of $m^\star$, send the message $(\texttt{OK}, m^\star)$ to all the parties, output $m^\star$ and terminate the protocol.

Figure 7.1: Bracha's asynchronous broadcast protocol tolerating $t < n/3$ corruptions [49]

Theorem 7.1 ([49]) *Protocol* a-cast *is an asynchronous broadcast protocol.*

7.2.3.2 Online Error Correction (oec) [49, 20]

Online error correction protocol oec can be viewed as the method of applying Reed-Solomon (RS) error-correction [154] in the asynchronous setting. Given a value which is t-shared among a set of parties $\mathcal{P}$, the goal is to publicly reconstruct the value robustly (actually oec allows

the parties to reconstruct the entire polynomial using which the value is t-shared). In the synchronous setting, this can be achieved by asking every party in $\mathcal{P}$ to send its share. Then the parties apply RS error-correction to reconstruct the value. Given the condition $t < (|\mathcal{P}|-2t)$, the reconstruction will be robust. In the asynchronous setting, achieving the same goal requires a small trick.

The intuition behind oec is that the parties wait till they receive the shares of $2t+1$ parties, all of which lie on a unique polynomial of degree t. This step requires applying RS error-correction repeatedly. We denote an RS error-correcting procedure as RS-dec(t, r, W) that takes as input a vector W of shares (some of them possibly incorrect) of a t-shared value (that we would like to reconstruct) and tries to output a polynomial of degree t, by correcting at most r errors in W. Coding theory [154] says that RS-dec can correct r errors in W and correctly interpolate the original polynomial provided that $|W| \geq t + 2r + 1$. There are several efficient implementations of RS-dec (for example, the Berlekamp-Welch algorithm [154]). Once the parties receive the shares from $2t+1$ parties that lie on a unique polynomial of degree t (returned by RS-dec), then that unique polynomial is the actual polynomial, say $Q(x)$, of degree t that defines t-sharing of $Q(0)$. This is because at least $t+1$ values out of the $2t+1$ values are from the honest parties, which uniquely define the original polynomial $Q(x)$. Note that the corrupted parties in $\mathcal{P}$ may send wrong values. But there are at least $n - t \geq 2t + 1$ honest parties in the set $\mathcal{P}$ whose shares will be eventually received by all parties in $\mathcal{P}$ and so every honest party in $\mathcal{P}$ will eventually terminate the process. The above procedure is nothing but applying RS error-correction algorithm in an "online" fashion.

The steps for the oec are now presented in Figure 7.2. The current description is inspired from [49]. The properties of oec are stated in the following theorem.

Protocol oec()

Setting: A value is t-shared among a set of parties $\mathcal{P}$, where $Q(x)$ is the underlying sharing polynomial of degree at most t, where $t < (n-2t)$; each party $P_i \in \mathcal{P}$ holds the share $v_i = Q(i)$ of $Q(0)$. The parties are supposed to publicly reconstruct the polynomial $Q(x)$.

Code for the party P_i — Every party in $\mathcal{P}$ executes the following code:

1. Send share v_i to all $P_j \in \mathcal{P}$.
2. For $r = 0, \ldots, t$, do the following in iteration r:
   (a) Let $\mathcal{W}$ denote the set of parties in $\mathcal{P}$ from whom P_i has received the shares and I_r denote the values received from the parties in $\mathcal{W}$, when $\mathcal{W}$ contains exactly $2t + 1 + r$ parties.

(b) Wait until $|\mathcal{W}| \geq 2t + 1 + r$. Execute $\mathsf{RS\text{-}dec}(t, r, I_r)$ to get a polynomial $Q_r(x)$ of degree t. If no polynomial is output, then skip the next step and proceed to the next iteration.

(c) If for at least $2t + 1$ values $v_j \in I_r$ it holds that $Q_r(j) = v_j$, then output $Q_r(x)$ and terminate. Otherwise, proceed to the next iteration.

Figure 7.2: Protocol for online error-correction [49]

Theorem 7.2 ([49, 20]) *Let a value be t-shared among a set of parties $\mathcal{P}$ where $t < (|\mathcal{P}| - 2t)$ and let $Q(x)$ be the underlying sharing polynomial. Then protocol* oec *achieves the following properties for every possible $\mathcal{A}$:*

– **Termination:** *Every honest party in $\mathcal{P}$ eventually terminates the protocol.*

– **Correctness:** *Each honest party in $\mathcal{P}$ upon terminating outputs $Q(x)$.*

7.2.4 Beaver's Circuit Randomization Technique

Beaver's circuit randomization method [18] is a well known method for securely computing $[x \cdot y]$, from $[x]$ and $[y]$, using a *precomputed* t-shared random and private multiplication-triple, say $([a], [b], [c])$, at the expense of two *public reconstructions* of t-shared values. For this, the parties first (locally) compute $[e]$ and $[d]$, where $[e] = [x] - [a] = [x - a]$ and $[d] = [y] - [b] = [y - b]$, followed by the public reconstruction of $e = (x - a)$ and $d = (y - b)$. Since the relation $xy = ((x - a) + a)((y - b) + b) = de + eb + da + c$ holds, the parties can locally compute $[xy] = de + e[b] + d[a] + [c]$, once d and e are publicly known. The above computation leaks no additional information about x and y if a and b are random and unknown to the adversary. We present protocol Beaver in Figure 7.3.

Protocol Beaver()

– **Input of the parties in $\mathcal{P}$:** A t-shared multiplication triple $([a], [b], [c])$ and t-shared pair $([x], [y])$.

– **Primitives Used:** Protocol oec

– **The Protocol:** The protocol runs asynchronously.

1. Compute $[e]$ and $[d]$ as $[e] = [x] - [a] = [x - a]$ and $[d] = [y] - [b] = [y - b]$.
2. The parties publicly reconstruct d and e using two instances of oec.
3. The parties locally compute $[xy] = de + e[b] + d[a] + [c]$.

Figure 7.3: Beaver's Circuit Randomization for Multiplication

The properties of the protocol Beaver are stated in Theorem 7.3.

Theorem 7.3 *([18]) Given a t-shared multiplication triple $([a], [b], [c])$ and t-shared pair $([x], [y])$ unknown to the adversary $\mathcal{A}$, Protocol* Beaver *achieves the following for every possible $\mathcal{A}$:*

– **Termination:** *All honest parties eventually terminate the protocol.*

– **Correctness:** *The honest parties correctly output $[xy]$ at the end of the protocol.*

– **Privacy:** *The view of $\mathcal{A}$ is distributed independently of the x and y values.*

7.2.5 Properties of Bivariate Polynomials

We state some well known standard properties of symmetric bivariate polynomials below from [9] which are used to prove security of our AVSS protocol.

Lemma 7.1 *([9]) Let $K \subseteq [n]$ be a set of indices such that $|K| \geq t+1$. Let $\{f_k(x)\}_{k \in K}$ be a set of polynomials of degree atmost t. If for every $i, j \in K$, it holds that $f_i(j) = f_j(i)$, then there exists a unique symmetric bivariate polynomial $F(x, y)$ of degree atmost t such that $f_i(x) = F(x, i) = F(i, x)$ for every $i \in K$.*

Lemma 7.2 *([9]) Suppose $I \subset \{1, \ldots, n\}$ with $|I| \leq t$, and $q_1(x)$, $q_2(x)$ are two degree t polynomials over $\mathbb{F}$ such that $q_1(i) = q_2(i)$ for every $i \in I$. Then the following distributions are indistinguishable; i.e,*

$$\{(i, F^1(x, i))\}_{i \in I}\} \equiv \{(i, F^2(x, i))\}_{i \in I}$$

where $F^1(x, y)$ and $F^2(x, y)$ are symmetric degree t bivariate polynomials chosen at random under the constraints that $F^1(x, 0) = q_1(x)$ and $F^2(x, 0) = q_2(x)$, respectively.

7.3 Asynchronous Weak Polynomial Sharing

We introduce a new primitive termed as *Asynchronous Weak Polynomial Sharing (AWPS)*. Informally the goal of the primitive is to allow a special party dealer $\mathsf{D} \in \mathcal{P}$ to share a random polynomial, say $f(x)$ over $\mathbb{F}$ of degree at most t among n parties in $\mathcal{P}$ so that every honest party P_i obtains ith point on the polynomial $f(i)$. When the dealer is corrupted, we wish to settle for a weaker requirement where corrupt D may choose a subset of honest parties to own shares of a unique and fixed polynomial of its choice. However, the subset must be big

enough, namely of size at least $t+1$, for D to be qualified in the AWPS. The most interesting and important property of AWPS is that every honest party including those who are outcasted will know whether it holds the share of the unique polynomial of corrupt D's choice and will terminate either with the correct share or with $\perp$.

An AWPS can be viewed as a primitive that allows to do 'weak' t-sharing (cf. Definition 7.3) of the constant term of the input polynomial. Specifically, when D is honest, AWPS outputs t-sharing of $f(0)$. When D is corrupt, it is guaranteed that at least a set of $t+1$ honest parties receive shares of $\bar{f}(0)$ where $\bar{f}(x)$ denotes the unique polynomial of degree at most t of D's choice. The remaining honest parties output $\perp$. We prefer AWPS as a primitive that shares a polynomial instead of a secret. The former representation fits better when the primitive is invoked in the AVSS protocol presented in the next section.

In the literature *Asynchronous Weak Secret Sharing* (AWSS) has been the key primitive that AVSS relied on. AWSS, much like AVSS, is a two-protocol $(\mathsf{Sh}, \mathsf{Rec})$ primitive and allows reconstruction of D's committed secret via Rec protocol in a 'weak sense' (reconstructs either the committed secret or $\perp$). Our AWPS primitive is weaker than AWSS in the sense that it only involves the sharing phase and may not even allow weak reconstruction of the committed secret. Yet, it helps build an AVSS protocol that achieves $2d$-sharing of a secret (cf. Definition 7.3), a lucrative feature that is very useful for building MPC. The formal definition of AWPS is presented below.

Definition 7.7 ***Asynchronous Weak Polynomial Sharing (AWPS):*** *A protocol* wsh *for n parties $\mathcal{P}$, where a dealer $\mathsf{D} \in \mathcal{P}$ holds a private polynomial $f(x)$ of degree at most t picked uniformly at random over $\mathbb{F}$ is a AWPS protocol tolerating $\mathcal{A}$ if the following requirements hold for every possible $\mathcal{A}$ and for all possible inputs of* D*:*

– **Termination***:*

1. *If* D *is honest and all the honest parties participate in the protocol* wsh*, then each honest party eventually terminates the protocol* wsh*.*
2. *If some honest party terminates* wsh*, then each honest party eventually terminates* wsh*.*

– **Correctness***: If some honest party terminates* wsh*, then there exists a fixed polynomial $\bar{f}(x)$ over $\mathbb{F}$, such that the following requirements hold. We refer this as '*D *has* partially-shared *$\bar{f}(x)$'.*

1. *If* D *is honest, then $\bar{f}(x) = f(x)$ and each honest party outputs $f(i)$.*

2. *If* D *is corrupt, then every honest party* P_i *upon completing* wsh *outputs either* $\bar{f}(i)$ *or* $\perp$. *Moreover, there exist a set of at least* $t+1$ *honest parties* $\mathcal{H}$ *such that each honest party* $P_i \in \mathcal{H}$ *upon completing* wsh *outputs* $\bar{f}(i)$.

– **Privacy**: *If* D *is honest then* $\mathcal{A}$*'s view is identically distributed for all possible* $f(0)$.

7.3.1 An AWPS Protocol in Hybrid network with One Synchronous Round

We present an AWPS protocol wsh with n parties $\mathcal{P}$ where $n \geq 3t+1$. The dealer initiates the protocol by picking a *symmetric* bivariate polynomial $F(x,y)$ of degree t in both variables uniformly at random over $\mathbb{F}$ such that $F(x,0)$ and $F(0,y)$ are the same as the input polynomial $f(x)$ (with change of variable for $F(0,y)$). Following some of the existing VSS protocols based on bivariate polynomials [139], the synchronous round of the protocol goes as follows: D sends $f_i(x) = F(x,i)$ to party P_i and in parallel the parties exchange random pads to be used for pairwise consistency checking of their common shares in the asynchronous phase. When a bivariate polynomial is distributed as above, a pair of parties (P_i, P_j) will hold the common share $F(i,j)$ via their respective polynomials $f_i(x)$ and $f_j(x)$. Namely, $f_i(j) = f_j(i) = F(i,j)$. In addition to the above information exchange, a party also registers the pads it sends to other parties with the dealer. Looking ahead, the dealer will use these pads in the asynchronous phase to detect potential conflicting pairs of parties.

In the asynchronous phase, every P_i sends its list of pads received during the synchronous round to the dealer. Using the sent list of pads from the synchronous round and the received list of pads as reported by P_i, the dealer records in a list c_i the set of parties who are conflicting with P_i and communicates the list to P_i. P_i publicly discloses its share in clear that is common with a party in conflict. For the rest, it makes the padded shares publicly available. D concludes a party to be *correct* if he finds that the padded values and the shares in clear are consistent with his chosen polynomial $F(x,y)$ and his recorded pad lists. D waits until he finds a set of at least $2t+1$ *correct* parties. D publicly announces the set denoted as $\mathcal{W}$. For an honest D, we show that any pair of parties in $\mathcal{W}$ will be *pairwise consistent*. To prevent a corrupt D from cheating, every honest party checks the sanity of $\mathcal{W}$ by enforcing that no pair in $\mathcal{W}$ publicly conflicts over their values that are either in padded or in clear form. So even a corrupted D cannot cheat and make the honest parties accept a set of (honest) parties who are not *pairwise consistent*. Once $\mathcal{W}$ is agreed upon, there is a unique bivariate polynomial, say $\bar{F}(x,y)$ that is committed by the dealer and is defined by the $f_i(x)$ polynomials of the honest parties in $\mathcal{W}$. The unique bivariate in turn defines a unique univariate polynomial $\bar{f}(x) = \bar{F}(x,0) = \bar{F}(0,y)$.

The honest parties in $\mathcal{W}$ outputs the constant term of their $f_i(x)$ polynomials received from the dealer as the share of $\bar{f}(x)$. The remaining parties who lie outside $\mathcal{W}$ compute their share by interpolating a degree t polynomial over its common shares with the parties in $\mathcal{W}$. If it has conflict with a party in $\mathcal{W}$, then the common share is publicly available. Otherwise, it computes the common share by subtracting the padded value made public by the party in $\mathcal{W}$ from the pad it sent in the synchronous round. For an honest D, the interpolation will result in a degree t polynomial $f_i(x)$ consistent with the dealer's original pick $F(x, y)$. Otherwise, the interpolation may fail and P_i outputs $\perp$. wsh is described in Figure 7.4.

For public announcement in asynchronous phase, we use the well-known asynchronous broadcast primitive known as a-cast, which was introduced and elegantly implemented by Bracha [43]. a-cast allows a sender in $\mathcal{P}$ to send a message identically to all the parties with a termination guarantee when the sender is honest. If the sender is corrupt and one honest party terminates then eventually every honest party will terminate the protocol with the *same* output. The complete details of a-cast is given in Section 7.2.3.1. We say that 'P_i a-casts x' to denote that P_i acts as a sender to broadcast x in an instance of a-cast.

Protocol wsh()

– **Input of** D**:** A random polynomial $f(x)$ of degree at most t over $\mathbb{F}$.

– **Primitives Used:** Protocol a-cast (cf. Section 7.2.3.1)

– **The Protocol:** It assumes a synchronous phase with one round followed by an asynchronous phase.

Synchronous Phase:

1. D chooses a random *symmetric* bivariate polynomial $F(x, y)$ of degree at most t in both variables such that $F(x, 0) = f(x)$. D sends the polynomial $f_i(x) = F(x, i)$ to party P_i.

2. Each party $P_i \in \mathcal{P}$ picks a random pad m_{ij} for every P_j and sends m_{ij} to P_j and D. D denotes the set of pads sent by P_i to other parties as $\{m^s_{ij}\}_{P_j \in \mathcal{P}}$.

Asynchronous Phase:

1. Each P_i sends its received list of pads $\{m_{ji}\}_{P_j \in \mathcal{P}}$ to D.

2. D denotes P_i's list as $\{m^r_{ji}\}_{P_j \in \mathcal{P}}$. It computes and sends to P_i a set c_i of all $P_j \in \mathcal{P}$ for which $m^r_{ji} \neq m^s_{ji}$.

3. Each P_i computes two lists $\mathcal{A}_i$ and $\mathcal{B}_i$ and a-casts $(\mathcal{A}_i, \mathcal{B}_i, \mathrm{c}_i)$.

- $\mathcal{A}_i = \{a_{ij} = f_i(j) + m_{ij}\}_{P_j \in \mathcal{P}}$
- $\mathcal{B}_i = \{b_{ij}\}_{P_j \in \mathcal{P}}$ where $b_{ij} = f_i(j)$ if $P_j \in \mathrm{c}_i$ and $b_{ij} = f_i(j) + m_{ji}$ otherwise.

4. On receiving $(\mathcal{A}_i, \mathcal{B}_i, \mathrm{c}_i)$, D marks a party P_i as *correct*, includes it in a set $\mathcal{W}$ and a-casts $\mathtt{Agree}_i$ if:
   - $a_{ij} - m^s_{ij} = F(i,j)$
   - $b_{ij} = F(i,j)$ for $\forall P_j \in \mathrm{c}_i$ and $b_{ij} - m^r_{ji} = F(i,j)$ otherwise
   - c_i is the same list that D communicated to P_i over point-to-point channel

   D waits until $|\mathcal{W}| \geq 2t+1$ and then a-casts $\mathcal{W}$.
5. On receiving $\mathcal{W}$, P_i accepts $\mathcal{W}$ if $\mathcal{W}$ is *valid* where a valid $\mathcal{W}$ satisfies the following:
   - For each $P_j \in \mathcal{W}$, $\mathtt{Agree}_j$ and $(\mathcal{A}_j, \mathcal{B}_j, \mathrm{c}_j)$ is received from the a-cast of D and P_j respectively
   - Every pair $(P_j, P_k) \in \mathcal{W}$ is *pairwise consistent* where pairwise consistency is defined as follows:
     i. if $(P_j \in \mathrm{c}_k \wedge P_k \in \mathrm{c}_j)$ then $b_{jk} = b_{kj}$,
     ii. if $(P_j \in \mathrm{c}_k \wedge P_k \notin \mathrm{c}_j)$ then $a_{kj} = b_{jk}$,
     iii. if $(P_j \notin \mathrm{c}_k \wedge P_k \in \mathrm{c}_j)$ then $a_{jk} = b_{kj}$,
     iv. else $a_{jk} = b_{kj}$ and $a_{kj} = b_{jk}$.
   - $|\mathcal{W}| \geq 2t+1$
6. On accepting $\mathcal{W}$, P_i outputs s_i and terminates wsh where s_i is computed as follows:
   - If $P_i \in \mathcal{W}$, then $s_i = f_i(0)$.
   - Else $s_i = \mathsf{constantTerm}(\{(j, s_{ij})\}_{P_j \in \mathcal{W}})$ where $s_{ij} = b_{ji}$ if $P_i \in \mathrm{c}_j$ and $s_{ij} = b_{ji} - m_{ij}$ otherwise.

Figure 7.4: A Weak Polynomial Sharing Protocol

Claim 7.1 *An honest* D *eventually consider every honest party* $P_i \in \mathcal{P}$ *as* correct *and* a-cast $\mathtt{Agree}_i$.

Proof: As both P_i and D are honest, all the following conditions will be true: (i) $a_{ij} - m^s_{ij} = F(i,j)$ (ii) $b_{ij} = F(i,j)$ for $\forall P_j \in \mathrm{c}_i$ and $b_{ij} - m^r_{ji} = F(i,j)$ otherwise (iii) c_i is the same list that D communicated to P_i over point-to-point channel. Hence the claim follows. □

Claim 7.2 *For an honest* D*, any pair of* correct *parties* (P_j, P_k) *is* pairwise consistent.

Proof: We consider the checks for the four possible cases of *pairwise consistency* checking based on whether P_j belongs to P_k's conflict list c_k and vice versa.

– $P_j \in \mathrm{c}_k \wedge P_k \in \mathrm{c}_j$: An honest D verifies that $b_{jk} = F(k, j)$ and $b_{kj} = F(j, k)$. So $b_{jk} = b_{kj}$ will hold.

– $P_j \in \mathrm{c}_k \wedge P_k \notin \mathrm{c}_j$: An honest D verifies that $a_{kj} = F(j, k) + m^s_{kj}$, $b_{jk} = F(j, k) + m^r_{kj}$ and $m^r_{kj} = m^s_{kj}$. The last equality follows from $P_k \notin \mathrm{c}_j$. So $a_{kj} = b_{jk}$ will hold.

– $P_j \notin \mathrm{c}_k \wedge P_k \in \mathrm{c}_j$: An honest D verifies that $a_{jk} = F(j, k) + m^s_{jk}$, $b_{kj} = F(j, k) + m^r_{jk}$ and $m^r_{jk} = m^s_{jk}$. The last equality follows from $P_j \notin \mathrm{c}_k$. So $a_{jk} = b_{kj}$ will hold.

– $P_j \notin \mathrm{c}_k \wedge P_k \notin \mathrm{c}_j$: An honest D verifies that $a_{jk} = F(j, k) + m^s_{jk}$, $b_{jk} = F(j, k) + m^r_{kj}$, $a_{kj} = F(j, k) + m^s_{kj}$, $b_{kj} = F(j, k) + m^r_{jk}$, $m^r_{jk} = m^s_{jk}$ and $m^r_{kj} = m^s_{kj}$. So $a_{jk} = b_{kj}$ and $a_{kj} = b_{jk}$.

The proof holds good irrespective of whether P_j and P_k are honest or corrupt. □

Claim 7.3 *If an honest P_i finds a set $\mathcal{W}$* valid*, then every other honest party will find it* valid *eventually.*

Proof: We note that the $\mathcal{W}$ set is a-casted by D. By the correctness property of a-cast, if honest P_i receives a set $\mathcal{W}$, all the other honest parties will receive the same set eventually. Since the validity checking of $\mathcal{W}$ is done on the information received from a-cast of various parties, every honest party will find the conditions true following the correctness property of a-cast. □

Lemma 7.3 *Protocol* wsh *satisfies the termination property stated in Definition 7.7.*

Proof: We first show that if D is honest and all the honest parties participate in protocol wsh, then each honest party eventually terminates the protocol. To prove the statement, we show that an honest D can always find a set $\mathcal{W}$ that will be accepted by one honest party. Subsequently, by Claim 7.3, every other honest party will accept $\mathcal{W}$. After accepting $\mathcal{W}$, the parties terminate protocol wsh after a local computation step. Now our claim that an honest D will always find a valid $\mathcal{W}$ where every pair will be *pairwise consistent* follows from the following argument. First, D will eventually consider every honest P_i as *correct* and a-cast $\mathtt{Agree}_i$ (by Claim 7.1) and there are at least $2t + 1$ honest parties. Second, any pair (P_i, P_j) who are considered to be *correct* by an honest D, no matter if they are honest or corrupt, they are *pairwise consistent* (by Claim 7.2).

Next, we show that if some honest party P_i terminates wsh, then each honest party eventually terminates wsh. If P_i terminates wsh, then it must have accepted a set $\mathcal{W}$ and subsequently

computed the output via local computation. By Claim 7.3, every other honest party will agree on the same $\mathcal{W}$ and will terminate the protocol after doing the local computation. □

Next, we prove the correctness property of wsh via a set of claims.

Claim 7.4 *If some honest party accepts a set $\mathcal{W}$, then the following hold:*

(a) *Each pair of honest parties (P_i, P_j) in $\mathcal{W}$ holds $\bar{f}_i(x)$ and $\bar{f}_j(x)$ respectively such that $\bar{f}_i(j) = \bar{f}_j(i)$.*

(b) *There exists a unique symmetric bivariate polynomial of degree at most t, say $\bar{F}(x, y)$ such that an honest P_i in $\mathcal{W}$ holds $\bar{f}_i(x)$ that is same as $\bar{F}(x, i)$. In case D is honest, $\bar{F}(x, y) = F(x, y)$.*

(c) *There exists a unique polynomial of degree at most t, say $\bar{f}(x)$ such that an honest P_i in $\mathcal{W}$ holds $\bar{f}(i)$. In case D is honest, $\bar{f}(x) = f(x)$.*

Proof: We prove the first part of the claim by contradiction. Assume a pair of honest parties $(P_i, P_j) \in \mathcal{W}$ such that $\bar{f}_i(j) \neq \bar{f}_j(i)$. We show that no honest party will accept $\mathcal{W}$ since (P_i, P_j) will not be *pairwise consistent*. Thus, we arrive at a contradiction.

- $P_i \in \mathrm{c}_j \wedge P_j \in \mathrm{c}_i$: Here $b_{ij} = \bar{f}_i(j)$ and $b_{ji} = \bar{f}_j(i)$. So $b_{ij} \neq b_{ji}$.
- $P_i \in \mathrm{c}_j \wedge P_j \notin \mathrm{c}_i$: Here $a_{ji} = \bar{f}_j(i) + m_{ji}$ and $b_{ij} = \bar{f}_i(j) + m_{ji}$. So $a_{ji} \neq b_{ij}$.
- $P_i \notin \mathrm{c}_j \wedge P_j \in \mathrm{c}_i$: Here $a_{ij} = \bar{f}_i(j) + m_{ij}$ and $b_{ji} = \bar{f}_j(i) + m_{ij}$. So $a_{ij} \neq b_{ji}$.
- $P_i \notin \mathrm{c}_j \wedge P_j \notin \mathrm{c}_i$: Here both $a_{ij} \neq b_{ji}$ and $a_{ji} \neq b_{ij}$.

For the second part, we note that the number of honest parties in $\mathcal{W}$ is at least $t + 1$ (since $|\mathcal{W}| \geq 2t + 1$). By the property of bivariate polynomials (Lemma 7.1) of degree at most t in both variables, there exist a unique bivariate polynomial say $\bar{F}(x, y)$ such that for every honest $P_i \in \mathcal{W}$ it holds that $\bar{f}_i(x) = \bar{F}(x, y)$. For an honest D, every honest $P_i \in \mathcal{W}$ holds the polynomial $f_i(x) = F(x, i)$ as given by the honest D in the synchronous round. Thus, the bivariate polynomial determined by the honest parties in $\mathcal{W}$ is $F(x, y)$.

For the last part, we define the unique polynomial $\bar{f}(x)$ as $\bar{F}(x, 0)$ where $\bar{F}(x, y)$ is as defined in the proof of part **(b)**. By part **(b)**, every honest P_i in $\mathcal{W}$ holds $\bar{f}_i(x) = \bar{F}(x, y)$. To complete the proof, we claim that $\bar{f}_i(0)$ is same as $\bar{f}(i)$. The equality $\bar{f}(i) = \bar{f}_i(0)$ holds following the given equalities: (i)$\bar{F}(x, 0) = \bar{f}(x)$ and $\bar{F}(i, 0) = \bar{f}(i)$ (the evaluation of the previous equation at 0), (ii) $\bar{f}_i(x) = \bar{F}(x, i)$ (by the definition of $\bar{f}_i(x)$ polynomial) and $\bar{f}_i(0) = \bar{F}(0, i)$ (the evaluation of the previous equation at 0) and (iii) finally $\bar{F}(i, 0) = \bar{F}(0, i)$ (by the symmetricity of $\bar{F}(x, y)$).

It is easy to note that in case D is honest, $\bar{f}(x) = f(x)$, since $\bar{F}(x, y) = F(x, y)$ as proved in part **(b)**. □

Lemma 7.4 *Protocol* wsh *satisfies the correctness property stated in Definition 7.7.*

Proof: For an honest dealer D, we show that every honest party P_i outputs $f(i)$ upon termination where $f(x)$ is the input of D in the protocol. Part **(c)** of Claim 7.4 shows that every honest $P_i \in \mathcal{W}$ outputs $f(i)$. To conclude the honest D case, we show that the above is true for every honest $P_i \notin \mathcal{W}$. Recall that P_i computes a set of values $\{s_{ij}\}_{P_j \in \mathcal{W}}$ such that $s_{ij} = b_{ji}$ if $P_i \in \mathrm{c}_j$ and $s_{ij} = b_{ji} - m_{ij}$ otherwise and finally computes $f(i)$ as $\mathsf{constantTerm}(\{(j, s_{ij})\}_{P_j \in \mathcal{W}})$. When the dealer is honest, $b_{ji} = f_j(i) = f_i(j)$ if $P_i \in \mathrm{c}_j$ and $b_{ji} - m_{ij} = f_j(i) = f_i(j)$ otherwise. Therefore, we have $\{(j, s_{ij})\}_{P_j \in \mathcal{W}} = \{(j, f_j(i))\}_{P_j \in \mathcal{W}} = \{(j, f_i(j))\}_{P_j \in \mathcal{W}}$. The set of points is t-consistent and passes through $f_i(x)$. It follows now that s_i as returned by $\mathsf{constantTerm}$ is same as $f_i(0)$. Finally $f_i(0)$ is same as $f(i)$ as proved in part **(c)** of Claim 7.4.

For a corrupted D, we show that there exists a unique polynomial $\bar{f}(x)$ so that every honest P_i outputs $\bar{f}(i)$ or $\perp$ and there exists a set of at least $t + 1$ honest parties who outputs shares of $\bar{f}(x)$. By Claim 7.4, there exist unique polynomials of degree at most t, $\bar{F}(x, y)$ and $\bar{f}(x)$ such that $\bar{F}(x, 0) = \bar{f}(x)$ and an honest P_i in $\mathcal{W}$ holds $\bar{f}_i(x)$ and $\bar{f}(i)$. We show that $\bar{f}(x)$ is the unique polynomial so that every honest party will output either $\bar{f}(i)$ or $\perp$. Part **(c)** of Claim 7.4 shows that every honest $P_i \in \mathcal{W}$ outputs $\bar{f}(i)$. Here we show that $P_i \notin \mathcal{W}$ will either output either $\bar{f}(i)$ or $\perp$. Recall that P_i computes a set of values $\{s_{ij}\}_{P_j \in \mathcal{W}}$ such that $s_{ij} = b_{ji}$ if $P_i \in \mathrm{c}_j$ and $s_{ij} = b_{ji} - m_{ij}$ otherwise. We show that $\mathsf{constantTerm}(\{j, s_{ij}\}_{P_j \in \mathcal{W}})$ either returns $\bar{f}_i(0)$ or $\perp$ where $\bar{f}_i(x) = \bar{F}(x, i)$. In other words, the points in $\{j, s_{ij}\}_{P_j \in \mathcal{W}}$ are either t-consistent and pass through $\bar{f}_i(x) = \bar{F}(x, i)$ or are not t-consistent. To prove the claim, we show that at least $t + 1$ s_{ij}'s are same as $\bar{f}_i(j)$. Specifically, we prove that $\{j, s_{ij}\}_{P_j \in \mathcal{H}} = \{j, \bar{f}_j(i)\}_{P_j \in \mathcal{H}} = \{j, \bar{f}_i(j)\}_{P_j \in \mathcal{H}}$ where $\mathcal{H}$ denotes the set of honest parties in $\mathcal{W}$. For an honest P_j, $s_{ij} = \bar{f}_j(i)$ no matter whether $P_i \in \mathrm{c}_j$ or not. Because, if $P_i \in \mathrm{c}_j$ then $s_{ij} = \bar{f}_j(i)$. Otherwise, it must be true that $b_{ji} = \bar{f}_j(i) + m_{ij}$ and so $s_{ij} = \bar{f}_j(i)$. The set of points are t-consistent and pass through $\bar{f}_i(x) = \bar{F}(x, i)$. Lastly $|\mathcal{H}|$ is at least $t + 1$ and every party in $\mathcal{H}$ outputs $\bar{f}(i)$. This concludes our correctness proof. □

Lemma 7.5 *Protocol* wsh *satisfies the privacy property stated in Definition 7.7.*

Proof: Let $\mathcal{C}$ denote the set of parties under control of $\mathcal{A}$ with $|\mathcal{C}| \leq t$. We first show that if D is honest, then the information the adversary has about the dealer's input at the end of wsh consists of the polynomials $\{f_i(x)\}_{P_i \in \mathcal{C}}$. We then argue that this information is independent of $f(0)$.

In the synchronous phase, $\mathcal{A}$ gets access to the polynomials $\{f_i(x)\}_{P_i \in \mathcal{C}}$. As for the values broadcast in the asynchronous phase, consider a pair of honest parties, say (P_i, P_j). Since P_i,P_j are honest, it must be true that $m^r_{ji} = m^s_{ji}$ and $m^r_{ij} = m^s_{ij}$. Consequently, $P_j \notin \mathrm{c}_i$ and $P_i \notin \mathrm{c}_j$. We now argue that the broadcast of P_i with respect to P_j i.e $a_{ij} = f_i(j) + m_{ij}$ and $b_{ij} = f_i(j) + m_{ji}$ will not leak anything about the value of $f_i(j) = F(i,j)$ to $\mathcal{A}$. This follows from the fact that m_{ij}, m_{ji} are chosen randomly and unknown to $\mathcal{A}$ since P_i, P_j, D are all honest. Similar argument holds for the broadcast of P_j in the asynchronous phase as well. Now suppose atleast one among (P_i, P_j) is corrupt, say P_j. Then it is possible that $P_i \in \mathrm{c}_j$ or $P_j \in \mathrm{c}_i$ and the point $F(i,j)$ may be broadcast in the asynchronous phase. However, this reveals no new information since $F(i,j) = f_i(j) = f_j(i)$ is already known to $\mathcal{A}$ controlling P_j.

We have seen above that the adversary's view consists nothing beyond the polynomials $\{f_i(x)\}_{P_i \in \mathcal{C}}$. We now argue this view is identically distributed for all possible $f(0)$. Consider any two degree-t polynomials $q_1(x)$ and $q_2(x)$ such that $q_1(i) = q_2(i) = f_i(0)$ for every $i \in \mathcal{C}$. Then, according to Lemma 7.2

$$\left\{(i, F^1(x,i))\}_{i \in \mathcal{C}}\right\} \equiv \left\{\{(i, F^2(x,i))\right\}_{i \in \mathcal{C}}$$

where $F^1(x,y)$ and $F^2(x,y)$ are symmetric degree t bivariate polynomials chosen at random under the constraints that $F^1(x,0) = q_1(x)$ and $F^2(x,0) = q_2(x)$, respectively. In more detail, the distribution over the shares $\{f_i(x)\}_{i \in \mathcal{C}}$ received by the corrupted parties when $F(x,y)$ is chosen based on $q_1(x)$ (i.e when $F(x,y) = F^1(x,y)$) is identical to the distribution when $F(x,y)$ is chosen based on $q_2(x)$ (i.e when $F(x,y) = F^2(x,y)$). In other words, no information is revealed about whether the private polynomial of D equals $q_1(x)$ or $q_2(x)$. Hence, we can conclude that $\{f_i(x)\}_{P_i \in \mathcal{C}}$ is independent of D's private polynomial $f(x)$, and consequently $f(0)$. □

Theorem 7.4 *Protocol* wsh *is a AWPS protocol.*

Proof: The theorem follows from Lemmas 7.3,7.4 and 7.5. □

7.4 Asynchronous VSS

In this section, we describe an AVSS protocol with n parties $\mathcal{P}$ where $n \geq 3t + 1$. In the Sh protocol, the dealer $2d$-shares (cf. Definition 7.3) its secret s via a symmetric bivariate polynomial $F(x,y)$ with its constant term as the secret. The secret s will be shared via $F(0,y)$ (which is same as $F(x,0)$), while the ith share of s, denoted as s_i and is same as $F(0,i)$ (and also $F(i,0)$) will be shared via $F(x,i)$. For the reconstruction, the parties participate in an

instance of online error correction protocol denoted as oec with the shares of s. Namely P_i participates in oec with s_i to reconstruct s. oec can be viewed as the method of applying Reed-Solomon (RS) error-correction [154] in the asynchronous setting that allows each party to robustly reconstruct not only the t-shared value but also the underlying polynomial used for sharing. The complete details of oec is given in Section 7.2.3.2. We now take a detailed look at the Sh protocol.

At a high level, Sh performs two layers of communication. The first layer of communication helps to determine a set of parties called $\mathcal{R}$ in which the information held by the set of honest parties uniquely determine a symmetric bivariate polynomial of degree t and hence determine a unique secret committed by the dealer. This layer resembles the wsh protocol structure where D picks a symmetric bivariate polynomial $F(x, y)$ of degree t so that its constant term equals the secret s. Then in the synchronous round, D hands out $f_i(x) = F(x, i)$ to party P_i and each pair of parties exchange random pads to check pairwise consistency of their common shares later in the asynchronous phase. The parties also record their sent and received set of pads with D. During the asynchronous phase, based on recorded pads D finds and publishes a set of at least $2t + 1$ *correct* parties, called $\mathcal{R}$ who are *pairwise consistent*. The first layer of communication seems to be not enough to achieve AVSS since the honest parties outside $\mathcal{R}$ may not hold a polynomial that is consistent with the unique committed bivariate polynomial in case of a corrupt dealer. This is exactly what happened in wsh and so at most t honest parties may end up with no share or $\perp$. The second layer of communication repairs the above drawback and aids every honest party outside $\mathcal{R}$ to recover a correct and consistent polynomial $f_i(x)$ and thus also a share of the committed secret. The second layer includes n instances of wsh led by the individual parties. Every honest party P_i *partially-shares* a random polynomial $p_i(x)$ via wsh_i, the ith instance of wsh and uses $p_i(x)$ to publicly commit its polynomial $f_i(x)$ in blinded form. Namely, P_i makes $f_i(x) + p_i(x)$ public. Later if honest P_i is declared to be part of $\mathcal{R}$, a party $P_j \notin \mathcal{R}$ can recover $f_i(j)$ as and when it computes $p_i(j)$ as the output in wsh_i. Note that when P_i is honest, then an honest P_j will recover $p_i(j)$ by the correctness of wsh. If a corrupted P_i is part of $\mathcal{R}$, we ensure that it must have made public commitment of correct $f_i(x)$ polynomial in the blinded form. This is enforced as follows. First, we ask each party P_j to conditionally participate in P_i's instance wsh_i based on whether the blinded polynomial made public by P_i is consistent with respect to its received point on $p_i(x)$ and common share $f_i(j)$. Second, P_i is part of $\mathcal{R}$ only when its instance of wsh_i leads to a *valid* $\mathcal{W}$ (cf.Figure 7.4 for the definition of a valid $\mathcal{W}$) that has at least $2t + 1$ parties in common with $\mathcal{R}$. Recall that a valid $\mathcal{W}$ in wsh ensures the honest parties in $\mathcal{W}$ defines a unique polynomial $p_i(x)$. Unfortunately, for a corrupted $P_i \in \mathcal{R}$, an honest P_j outside $\mathcal{W}$ may output $\perp$ instead of $p_i(j)$ at the end of wsh_i.

This is not a problem since $\mathcal{R}$ contains at least $t+1$ honest parties and P_j can recover at least $t+1$ points on $f_j(x)$ which are clearly enough to recover $f_j(x)$. The AVSS protocol appears in Figure 7.5. We proceed to prove the properties.

AVSS Protocol

Protocol Sh()

– **Input of** D**:** A secret $s \in \mathbb{F}$

– **Primitives Used:** Protocol wsh, a-cast (cf. Section 7.2.3.1)

– **The Protocol:** It assumes a synchronous phase with one round followed by an asynchronous phase.

Synchronous Phase:

1. D chooses a random symmetric bivariate polynomial $F(x,y)$ of degree at most t such that $F(0,0) = s$ and sends the polynomial $f_i(x) = F(x,i)$ to each party $P_i \in \mathcal{P}$.

2. Each $P_i \in \mathcal{P}$ picks a random pad m_{ij} for every P_j and sends m_{ij} to P_j and D. D denotes the set of pads sent by P_i to other parties as $\{m^s_{ij}\}_{P_j \in \mathcal{P}}$.

3. Each P_i picks a random polynomial $p_i(x)$ and executes the synchronous phase of wsh() as a dealer with input $p_i(x)$. We refer to this instance as wsh_i. P_i participates in synchronous phase of wsh_j for $\forall P_j$.

Asynchronous Phase:

1. Each P_i sends its received list of pads $\{m_{ji}\}_{P_j \in \mathcal{P}}$ to D. D denotes P_i's list as $\{m^r_{ji}\}_{P_j \in \mathcal{P}}$. It computes and sends to P_i a set c_i of all $P_j \in \mathcal{P}$ for which $m^r_{ji} \neq m^s_{ji}$.

2. Each P_i computes two lists $(\mathcal{A}_i, \mathcal{B}_i)$ as follows and a polynomial $d_i(x) = p_i(x) + f_i(x)$. It then a-casts $(\mathcal{A}_i, \mathcal{B}_i, \mathrm{c}_i, b_i(x))$.

   – $\mathcal{A}_i = \{a_{ij} = f_i(j) + m_{ij}\}_{P_j \in \mathcal{P}}$

   – $\mathcal{B}_i = \{b_{ij}\}_{P_j \in \mathcal{P}}$ where $b_{ij} = f_i(j)$ if $P_j \in \mathrm{c}_i$ and $b_{ij} = f_i(j) + m_{ji}$ otherwise.

3. Each P_i participates in wsh_i as the dealer. It participates in wsh_j if **(a)** $d_j(x)$ received from the a-cast of P_j is a polynomial of degree at most t and **(b)** $d_j(i) = p_j(i) + f_i(j)$.

4. D finds and A-casts sets $\mathcal{R} \subseteq \mathcal{P}$ and $\mathcal{W}_i \subseteq \mathcal{P}$, $\forall P_i \in \mathcal{R}$ with the following properties:

   – Every $P_i \in \mathcal{R}$ is *correct*

- $\forall P_i \in \mathcal{R}$, $\mathcal{W}_i$ is a *valid* $\mathcal{W}$ in wsh_i
- $|\mathcal{R}| \geq 2t+1$ and $|\mathcal{R} \cap \mathcal{W}_i| \geq 2t+1$

5. On receiving $(\mathcal{R}, \{\mathcal{W}_i\}_{P_i \in \mathcal{R}})$, P_i accepts the sets if the following conditions hold:
   - $\forall P_j \in \mathcal{R}$, $(\mathcal{A}_j, \mathcal{B}_j, \mathrm{c}_j)$ is received from the $\mathsf{a\text{-}cast}$ of P_j.
   - Every $(P_j, P_k) \in \mathcal{R}$ is *pairwise consistent*.
   - $\forall P_j \in \mathcal{R}$, $\mathcal{W}_j$ is a *valid* $\mathcal{W}$ in wsh_j.
   - $|\mathcal{R}| \geq 2t+1$ and $|\mathcal{R} \cap \mathcal{W}_i| \geq 2t+1$
6. On accepting the sets, P_i outputs s_i and terminates Sh where s_i is computed as follows:
   - If $P_i \in \mathcal{R}$, then it sets $s_i = f_i(0)$.
   - Otherwise, it computes its output p_{ij} in wsh_j for every $P_j \in \mathcal{R}$, finds the set $\{s_{ij} = d_j(i) - p_{ij}\}_{P_j \in \mathcal{R} \wedge p_{ij} \neq \perp}$ and outputs $s_i = \mathsf{constantTerm}(\{(j, s_{ij})\}_{P_j \in \mathcal{R} \wedge p_{ij} \neq \perp})$.

Protocol $\mathsf{Rec}()$

- **Input of each** $P_i \in \mathcal{P}$**:** s_i i.e P_i's output of Sh
- **Primitives Used:** Protocol oec (cf. Section 7.2.3.2)
- **The Protocol:** It assumes an asynchronous network.
  The parties run an instance of oec and terminate with $s = f(0)$ where $f(x)$ is the output of oec.

Figure 7.5: AVSS Protocol

Claim 7.5 *If* D *is honest, then he eventually finds sets* $(\mathcal{R}, \{\mathcal{W}_i\}_{P_i \in \mathcal{R}})$ *which some honest party accepts.*

Proof: We first show that D can compute sets $(\mathcal{R}, \{\mathcal{W}_i\}_{P_i \in \mathcal{R}})$ satisfying the following three conditions in polynomial time: (i) Every P_i is *correct*, (ii) $\forall P_i \in \mathcal{R}$, $\mathcal{W}_i$ is a *valid* $\mathcal{W}$ in wsh_i, (iii) $|\mathcal{R}| \geq 2t+1$ and $|\mathcal{R} \cap \mathcal{W}_i| \geq 2t+1$. In the second part of the proof, we will show that the sets satisfying the above conditions will be accepted by some honest party. D follows the following steps to find the sets $(\mathcal{R}, \{\mathcal{W}_i\}_{P_i \in \mathcal{R}})$:

- **Initialisation:** Initialise a dynamic set $\mathcal{T}$ to $\emptyset$. For every party P_i initialise a dynamic set $\mathcal{W}_i$ to $\emptyset$.

– **Update of $\mathcal{W}$ Sets:** Include every P_j in $\mathcal{W}_i$ for which $\texttt{Agree}_j$ is received from the a-cast of P_i in wsh_i. If $\mathcal{W}_i$ is not *valid*, then reset $\mathcal{W}_i$ to $\emptyset$.

– **Update of $\mathcal{T}$:** Include a party P_i in $\mathcal{T}$ if the following are true: (i) P_i is *correct* and (ii) $|\mathcal{W}_i| \geq 2t + 1$.

– **Finding Candidate Solution:** If there exists a subset $\mathcal{R}$ of $\mathcal{T}$ such that $|\mathcal{R} \cap \mathcal{W}_i| \geq 2t + 1$ is true for every $P_i \in \mathcal{R}$, then return $(\mathcal{R}, \{\mathcal{W}_i\}_{P_i \in \mathcal{R}})$ and stop. Else wait and verify if the above condition is true after every update of $\mathcal{T}$ or $\mathcal{W}_i$s.

It is easy to note that if at some point of time $\mathcal{T}$ contains all the honest parties and for every honest party P_i, $\mathcal{W}_i$ contains all the honest parties, then D can find a candidate solution $(\mathcal{R}, \{\mathcal{W}_i\}_{P_i \in \mathcal{R}})$. The following observations prove that eventually the above event will happen. First, every honest P_i will be considered as *correct* by an honest D following Claim 7.1. Second, every honest P_i will a-cast $\texttt{Agree}_j$ for every honest P_j eventually in wsh_i. Consider an honest P_j participating in wsh_i of an honest P_i. P_j will find $d_i(x)$ of degree at most t. In addition, the relation $d_i(j) = p_i(j) + f_j(i)$ will hold true. So P_j will continue to participate in wsh_i. So P_i will a-cast $\texttt{Agree}_j$ for P_j following Claim 7.1. Third, for an honest P_i, the set of parties for which P_i a-casts $\texttt{Agree}$ at any point of time will always constitute a *valid* $\mathcal{W}_i$.

Now moving on to the second part of the proof, the sets $(\mathcal{R}, \{\mathcal{W}_i\}_{P_i \in \mathcal{R}})$ computed as above and a-casted by D will be accepted by an honest P_i. The conditions checked by P_i and D on the sets are different with respect to the following: P_i checks if every pair $(P_j, P_k) \in \mathcal{R}$ is *pairwise consistent*; while D checks if the parties in $\mathcal{R}$ are *correct*. The proof follows from Claim 7.2 that shows that for an honest D, the check for *correctness* guarantees *pairwise consistency*. □

Claim 7.6 *If one honest party accepts the sets $(\mathcal{R}, \{\mathcal{W}_i\}_{P_i \in \mathcal{R}})$, then every other honest party accepts them.*

Proof: It is easy to note that the checks done by the parties are based on a-casted information. By the correctness of a-cast, all the honest parties will receive the same information and conclude the same. □

Lemma 7.6 *The pair of protocols $(\mathsf{Sh}, \mathsf{Rec})$ satisfies the termination property stated in Definition 7.1.*

Proof: Claim 7.5 and Claim 7.6 together imply that if D is honest and all the honest parties participate in Sh, then each honest party eventually terminates the protocol. Because once the parties accept the sets $(\mathcal{R}, \{\mathcal{W}_i\}_{P_i \in \mathcal{R}})$, they terminate after local computation. Next, Claim 7.6

implies that if some honest party terminates Sh, then each honest party eventually terminates Sh. Lastly, if all the honest parties participate in Rec, it follows from the termination property of oec (cf. Theorem 7.2) that each honest party eventually terminates Rec. □

Claim 7.7 *If some honest party accepts sets* $(\mathcal{R}, \{\mathcal{W}_i\}_{P_i \in \mathcal{R}})$*, then the following holds:*

(a) *Each pair of honest parties* (P_i, P_j) *in* $\mathcal{R}$ *holds* $\bar{f}_i(x)$ *and* $\bar{f}_j(x)$ *respectively such that* $\bar{f}_i(j) = \bar{f}_j(i)$.

(b) *There exists a unique symmetric bivariate polynomial of degree at most* t*, say* $\bar{F}(x, y)$ *such that an honest* P_i *in* $\mathcal{R}$ *holds* $\bar{f}_i(x)$ *that is same as* $\bar{F}(x, i)$*. In case* D *is honest,* $\bar{F}(x, y) = F(x, y)$.

(c) *A corrupt* $P_i \in \mathcal{R}$ *must have* a-cast*ed* $d_i(x) = \bar{F}(x, i) + p_i(x)$ *where* $p_i(x)$ *denotes the unique polynomial* P_i *has* partially-shared *in* wsh_i*. In case* D *is honest,* $\bar{F}(x, i) = F(x, i)$.

(d) *An honest* $P_i \notin \mathcal{R}$ *eventually holds* $\bar{f}_i(x)$ *that is same as* $\bar{F}(x, i)$*. In case* D *is honest,* $\bar{F}(x, i) = F(x, i)$.

Proof: The statements in **(a)** and **(b)** can be proved in the same way Claim 7.4 has been proved with respect to $\mathcal{W}$ of wsh. We now prove part **(c)**. By Lemma 7.4, every honest party P_j in $\mathcal{W}_i$ holds $p_i(j)$. Now consider the honest parties in $\mathcal{R} \cap \mathcal{W}_i$. Each such honest party P_j holds $\bar{f}_j(x)$ that is same as $\bar{F}(x, j)$ (by part **(b)**) and ensures that P_i's a-casted polynomial $d_i(x)$ evaluates to $\bar{f}_j(i) + p_i(j)$ and is of degree t. Since $\mathcal{R} \cap \mathcal{W}_i$ contains at least $t+1$ honest parties and $\bar{f}_j(i) = \bar{f}_i(j)$, $d_i(x)$ evaluates to the set of at least $t+1$ points $\{\bar{f}_i(j) + p_i(j)\}_{P_j \in \mathcal{R} \cap \mathcal{W}_i}$ corresponding to the honest parties in $\mathcal{R} \cap \mathcal{W}_i$. So we can conclude that $d_i(x)$ is identical to $\bar{F}(x, i) + p_i(x)$. It is easy to note that for an honest D, $\bar{F}(x, i) = F(x, i)$.

We now prove the last part. Consider an honest $P_i \notin \mathcal{R}$. For every honest $P_j \in \mathcal{R}$, P_i outputs p_{ij} that is same as $p_j(i)$ where $p_j(x)$ is *partially-shared* by honest P_j in wsh_j (by the correctness property of wsh). Therefore, for every honest P_j, P_i will recover $\bar{f}_j(i)$ (and so $\bar{f}_i(j)$) as $d_i(j) - p_{ij}$. For every corrupted $P_j \in \mathcal{R}$, P_i outputs either $\bar{f}_i(j)$ or $\perp$ based on whether p_{ij} is $p_j(i)$ or $\perp$, where $p_j(x)$ is *partially-shared* by corrupted P_j in wsh_j. The correctness of $\bar{f}_i(j)$ is guaranteed by part **(c)**. Since $\mathcal{R}$ contains at least $t+1$ honest parties, the set of points $\{s_{ij} = d_j(i) - p_{ij}\}_{P_j \in \mathcal{R} \wedge p_{ij} \neq \perp}$ is big enough and contains correct points on $\bar{f}_i(x)$. Therefore, P_i can recover $\bar{f}_i(x)$ as well as the constant term s_i of $\bar{f}_i(x)$ by running $s_i = \mathsf{constantTerm}\big(\{(j, s_{ij})\}_{P_j \in \mathcal{R} \wedge p_{ij} \neq \perp}\big)$. It is easy to note that for an honest D, $\bar{F}(x, i) = F(x, i)$. □

Lemma 7.7 *The pair of protocols* (Sh, Rec) *satisfies the correctness property stated in Definition 7.1*

Proof: From part **(b)** and **(d)** of Claim 7.7, there exists a symmetric bivariate polynomial of degree at most t, say $\bar{F}(x,y)$ such that each honest P_i holds $\bar{f}_i(x)$ that is same as $\bar{F}(x,i)$. We prove that $\bar{s} = \bar{F}(0,0)$ is the unique secret shared by D which will be reconstructed after running Rec protocol. Each party honest party holds $\bar{f}_i(x)$ and finally outputs $\bar{s}_i = \bar{f}_i(0)$ in Sh as the share of $\bar{s}$. Clearly, the $\bar{s}_i$ values define t-sharing of $\bar{s}$ via degree t polynomial $\bar{f}_0(x) = \bar{F}(x,0)$ (which is same as $\bar{F}(0,y)$ by the symmetry of the bivariate polynomial). Now by the correctness of oec (cf. Theorem 7.2), the parties will output $\bar{s}$ at the end of Rec. Finally, if D is honest $\bar{F}(x,y)$ is same as $F(x,y)$, the original pick of the honest D and so $s = \bar{s}$. □

Lemma 7.8 *The pair of protocols* $(\mathsf{Sh}, \mathsf{Rec})$ *satisfies the privacy property stated in Definition 7.1*

Proof: Let $\mathcal{C}$ denote the set of parties under control of $\mathcal{A}$ where $|\mathcal{C}| \leq t$. $\mathcal{A}$ gets access to $\{f_i(x)\}_{P_i \in \mathcal{C}}$ in the synchronous phase. In addition to this, $\mathcal{A}$ also knows the polynomial $d_i(x) = f_i(x) + p_i(x)$ broadcast by an honest P_i in the asynchronous phase. Let us now check if $d_i(x)$ leaks any information. Recall that P_i acts as dealer in wsh_i. Since P_i is honest, it follows from privacy of wsh_i (Lemma 7.5) that $p_i(x)$ remains unknown to $\mathcal{A}$. Therefore, $\mathcal{A}$ does not learn anything from the broadcast $d_i(x)$ since $f_i(x)$ is 'blinded' by the polynomial $p_i(x)$ unknown to $\mathcal{A}$. Also, by the same argument as in Lemma 7.5, broadcast of values by honest parties in the asynchronous phase will reveal no new information about the secret s. We can now conclude that $\mathcal{A}$ knows nothing beyond $\{f_i(x)\}_{P_i \in \mathcal{C}}$. We now argue that this information is independent of s. Consider any two degree-t polynomials $q_1(x)$ and $q_2(x)$ such that $q_1(i) = q_2(i) = f_i(0)$ for every $i \in \mathcal{C}$. Then, according to Lemma 7.2

$$\left\{(i, F^1(x,i))\right\}_{i \in \mathcal{C}} \equiv \left\{(i, F^2(x,i))\right\}_{i \in \mathcal{C}}$$

where $F^1(x,y)$ and $F^2(x,y)$ are symmetric degree t bivariate polynomials chosen at random under the constraints that $F^1(x,0) = q_1(x)$ and $F^2(x,0) = q_2(x)$, respectively. In more detail, the distribution over the shares $\{f_i(x)\}_{i \in \mathcal{C}}$ received by the corrupted parties when $F(x,y)$ is chosen based on $q_1(x)$ (i.e when $F(x,y) = F^1(x,y)$) is identical to the distribution when $F(x,y)$ is chosen based on $q_2(x)$ (i.e when $F(x,y) = F^2(x,y)$). An important point to be noted is that both $q_1(x)$ and $q_2(x)$ are consistent with the information $\{f_i(x)\}_{i \in \mathcal{C}}$ possessed by $\mathcal{A}$ since $q_1(i) = q_2(i) = f_i(0)$ for every $i \in \mathcal{C}$. Consequently, no information is revealed about whether the secret equals $s_1 = q_1(0)$ or $s_2 = q_2(0)$. Hence, we can conclude that $\{f_i(x)\}_{P_i \in \mathcal{C}}$ is independent of D's input s, which completes the proof. □

Theorem 7.5 *The pair of protocols* $(\mathsf{Sh}, \mathsf{Rec})$ *is an AVSS protocol.*

Proof: The theorem follows from Lemma 7.6,7.7 and 7.8. □

Theorem 7.6 *The value $\bar{s}$ committed by* D *during* Sh *is 2d-shared. If* D *is honest, then $\bar{s} = s$.*

Proof: It is easy to see that $\bar{s} = \bar{F}(0,0)$ is t-shared via $\bar{F}(0,y)$ and each $P_i \in \mathcal{P}$ holds $\bar{s}_i = \bar{f}_i(0) = \bar{F}(0,i)$ (Lemma 7.7). Now each $\bar{s}_i$ is t-shared via $\bar{f}_i(x)$ which is same as $\bar{F}(0,y)$. Because, each honest P_i holds $\bar{f}_j(x)$ and therefore $\bar{f}_i(j)$ (which is same as $\bar{f}_j(i)$). If D is honest, then $\bar{s} = s$ where s is D's input to Sh. □

7.5 Impossibility of AMPC with One Synchronous Round

In this section, we prove impossibility of a perfectly-secure AMPC (Definition 7.4) with $n \leq 4t$ parties over a network that provides a single synchronous round with broadcast oracle access prior to turning to asynchronous mode. The proof takes inspiration from the proof of impossibility of perfectly-secure AMPC with $n \leq 4t$ [49].

Theorem 7.7 *For every $n \geq 4$, there exist functions f such that no perfectly-secure AMPC protocol can compute f with $n \leq 4t$ parties over a hybrid network that supports a single synchronous round with broadcast oracle access.*

Proof: Consider the setting $(n = 4, t = 1)$. The proof can easily be generalized to all $n \leq 4t$. Let $\mathcal{P} = \{P_1, P_2, P_3, P_4\}$ denote the set of parties. We prove the theorem by contradiction. We assume that there exists a perfectly-secure AMPC protocol π which computes the function $f(x_1, x_2, x_3, x_4)$ defined below for P_i's input x_i over a hybrid network that provides a single synchronous round with access to broadcast oracle:

$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_2 = x_3 = 1 \\ 0 & \text{otherwise} \end{cases}$$

Protocol π consists of two sub-protocols or phases: synchronous phase is denoted as π_s and asynchronous phase is denoted as π_a. We write $\pi = (\pi_s, \pi_a)$. We assume that the communication done in the asynchronous phase in π are done via broadcast. This holds without loss of generality since the parties can perform point-to-point communication by exchanging random pads in the first round and then using these random pads to unmask later broadcasts [101].

We will consider an execution of π where $\mathcal{A}$ corrupts P_1 and follows a scheduling strategy and a strategy for P_1 in π_a as described below. In each step of π_a, $\mathcal{A}$ delivers the messages of parties P_1, P_2 and P_3 in 'round robin' fashion. The messages of P_4 are delivered only when there are no undelivered messages of the other parties. $\mathcal{A}$ communicates on behalf of P_1 in a

malicious way (details below) such that P_2 and P_3 do not identify P_1 as corrupt. So in each step P_2 and P_3 wait for messages from only $(n-t) = 4-1 = 3$ parties and proceed to the next step before any message from P_4 is delivered. This strategy of $\mathcal{A}$ ensures that P_2 and P_3 terminate π without any consideration of the communication from P_4 during π_a. The crux of the proof is to show that P_2 and P_3 will output 0 when π is run with input $(*, 1, 1, *)$ in the presence of the above scheduling strategy of $\mathcal{A}$ and a strategy for P_1 discussed below, where $*$ denotes any input value on behalf of P_1 and P_4. Since π is assumed to be a perfectly-secure AMPC, an error in correctness is not allowed. Therefore, we will conclude that π is not a perfectly-secure AMPC for f arriving at a contradiction.

$\mathsf{p}^{\pi_s}_{i\to j}$ denote the transcript of the point-to-point communication done in the first synchronous round from P_i to P_j and $\mathsf{p}^{\pi_s}_{i\leftrightarrow j}$ denote $\{\mathsf{p}^{\pi_s}_{i\to j}, \mathsf{p}^{\pi_s}_{j\to i}\}$. $\mathsf{b}^{\pi_s}_i$ and $\mathsf{b}^{\pi_a}_i$ denote the broadcast communication done by P_i in π_s and π_a respectively. We denote the view of party P_i at the end of π as V_i that constitutes of P_i's initial input x_i and random input r^π_i and the private and broadcast communication that it has received in π. The information sent out by P_i can be polynomially computed from the initial input and received information. Thereby, they are not considered as a part of the view. So the view of a party P_i can be defined as follows:

$$\mathsf{V}_i(\pi) = \left\{x_i, r^\pi_i, \{\mathsf{p}^{\pi_s}_{j\to i}\}_{j\neq i}, \{\mathsf{b}^{\pi_s}_j\}_{j\neq i}, \{\mathsf{b}^{\pi_a}_j\}_{j\neq i}\right\}$$

Let $\mathsf{T}^\pi_{i\to j}$ denote the transcript of the communication from P_i to P_j in π and $\mathsf{T}^\pi_{i\leftrightarrow j} = \{\mathsf{T}^\pi_{i\to j}, \mathsf{T}^\pi_{j\to i}\}$. We can write $\mathsf{T}^\pi_{i\to j} = \{\mathsf{p}^{\pi_s}_{i\to j}, \mathsf{b}^{\pi_s}_i, \mathsf{b}^{\pi_a}_i\}$ and $\mathsf{T}^\pi_{i\leftrightarrow j} = \{\mathsf{p}^{\pi_s}_{i\leftrightarrow j}, \mathsf{b}^{\pi_s}_i, \mathsf{b}^{\pi_s}_j, \mathsf{b}^{\pi_a}_i, \mathsf{b}^{\pi_a}_j\}$. Denoting $\pi(x_2, x_3)$ as an execution of π with the input of P_2 and P_3 as x_2 and x_3 respectively and $r^{\pi(x_2,x_3)}_i$ for $i \in [4]$ to denote the random inputs in execution $\pi(x_2, x_3)$, we prove the following useful claim:

Claim 7.8 *There exist* $\{r^{\pi(0,0)}_i\}_{i\in[4]}$, $\{r^{\pi(0,1)}_i\}_{i\in[4]}$, $\{r^{\pi(1,0)}_i\}_{i\in[4]}$ *and* $\{r^{\pi(1,1)}_i\}_{i\in[4]}$ *such that* $\mathsf{T}^{\pi(0,0)}_{2\leftrightarrow 3} = \mathsf{T}^{\pi(0,1)}_{2\leftrightarrow 3} = \mathsf{T}^{\pi(1,0)}_{2\leftrightarrow 3} = \mathsf{T}^{\pi(1,1)}_{2\leftrightarrow 3}$.

Proof: First, $\mathsf{T}^{\pi(0,0)}_{2\leftrightarrow 3} = \mathsf{T}^{\pi(0,1)}_{2\leftrightarrow 3}$ holds. If it is not true, then the transcript $\mathsf{T}^\pi_{2\leftrightarrow 3}$ carries information about P_3's input and so a corrupt P_2 can learn P_3's input. This will breach privacy since P_2 should learn nothing beyond the output 0. Second, $\mathsf{T}^{\pi(0,0)}_{2\leftrightarrow 3} = \mathsf{T}^{\pi(1,0)}_{2\leftrightarrow 3}$ holds. Otherwise, following similar argument as above, a corrupt P_3 can learn P_2's input. Finally, we argue that $\mathsf{T}^{\pi(0,1)}_{2\leftrightarrow 3} = \mathsf{T}^{\pi(1,1)}_{2\leftrightarrow 3}$ as follows. We claim that $\mathsf{T}^{\pi(0,1)}_{2\leftrightarrow 3}$ must be independent of x_3. Otherwise, corrupt P_2 can learn x_3, breaching P_3's privacy as P_2 should not learn anything beyond output 0. Next, since $\mathsf{T}^{\pi(0,1)}_{2\leftrightarrow 3}$ is independent of x_3, it must be consistent with $x_3 = 0$. Therefore, it must be such that it does not leak the input of P_2 to preserve P_2's privacy in case $x_3 = 0$. Thus, we

can conclude that $\mathsf{T}_{2\leftrightarrow 3}^{\pi(0,1)}$ is independent of P_2's input as well and $\mathsf{T}_{2\leftrightarrow 3}^{\pi(0,1)} = \mathsf{T}_{2\leftrightarrow 3}^{\pi(1,1)}$ holds. These equalities together prove the claim. □

We now consider two executions $\pi(0,1)$ and $\pi(1,0)$ and claim the following:

Claim 7.9 *There exist* $\{r_i^{\pi(0,1)}\}_{i\in[4]}$ *and* $\{r_i^{\pi(1,0)}\}_{i\in[4]}$ *such that:*

(a) $\mathsf{T}_{2\leftrightarrow 3}^{\pi(0,1)} = \mathsf{T}_{2\leftrightarrow 3}^{\pi(1,0)}$,

(b) $\{\mathsf{T}_{1\to 2}^{\pi(0,1)}, \mathsf{T}_{1\to 3}^{\pi(0,1)}, \mathsf{T}_{1\to 4}^{\pi(0,1)}\} = \{\mathsf{T}_{1\to 2}^{\pi(1,0)}, \mathsf{T}_{1\to 3}^{\pi(1,0)}, \mathsf{T}_{1\to 4}^{\pi(1,0)}\}$ *and*

(c) $\mathsf{p}_{4\to i}^{\pi_s(0,1)} = \mathsf{p}_{4\to i}^{\pi_s(1,0)}$ *and* $\mathsf{b}_4^{\pi_s(0,1)} = \mathsf{b}_4^{\pi_s(1,0)}$ *for* $i \in [3]$.

Proof: Part (a) follows from Claim 7.8. We now prove (b). If it is not true, then a corrupt P_1 can learn beyond the output 0, namely whether it is $(0,1)$ or $(1,0)$ that led to output which it is not supposed to learn. The third claim follows easily as the communication of P_4 in the synchronous round is only dependent on its own random input and therefore the equalities hold when $r_4^{\pi(0,1)} = r_4^{\pi(1,0)}$. □

Skipping the annotation of $(0,1)$ and $(1,0)$ for the communications that are equal across $\pi(0,1)$ and $\pi(1,0)$, we present the views honest P_1, P_2, P_3 and P_4 in these two executions in Table 7.2. We ignore the view and communication by P_4 beyond the synchronous round, as it never gets chance to participate in the execution, except the synchronous round as per the adversarial scheduling strategy mentioned above.

Table 7.2: Views of P_1, P_2, P_3, P_4 in $\pi(0,1)$ and $\pi(1,0)$

	Views in $\pi(0,1)$				Views in $\pi(1,0)$			
	$\mathsf{V}_1(\pi(0,1))$	$\mathsf{V}_2(\pi(0,1))$	$\mathsf{V}_3(\pi(0,1))$	$\mathsf{V}_4(\pi(0,1))$	$\mathsf{V}_1(\pi(1,0))$	$\mathsf{V}_2(\pi(1,0))$	$\mathsf{V}_3(\pi(1,0))$	$\mathsf{V}_4(\pi(1,0))$
Initial Input	$r_1^{\pi(0,1)}$	$(0, r_2^{\pi(0,1)})$	$(1, r_3^{\pi(0,1)})$	$r_4^{\pi(0,1)}$	$r_1^{\pi(1,0)}$	$(1, r_2^{\pi(1,0)})$	$(0, r_3^{\pi(1,0)})$	$r_4^{\pi(1,0)}$
π_s	$\mathsf{p}_{2\to 1}^{\pi_s(0,1)}$, $\mathsf{p}_{3\to 1}^{\pi_s(0,1)}$, $\mathsf{p}_{4\to 1}^{\pi_s}$ $\mathsf{b}_2^{\pi_s}, \mathsf{b}_3^{\pi_s}, \mathsf{b}_4^{\pi_s}$	$\mathsf{p}_{1\to 2}^{\pi_s}$, $\mathsf{p}_{3\to 2}^{\pi_s}$, $\mathsf{p}_{4\to 2}^{\pi_s}$ $\mathsf{b}_1^{\pi_s}, \mathsf{b}_3^{\pi_s}, \mathsf{b}_4^{\pi_s}$	$\mathsf{p}_{1\to 3}^{\pi_s}$, $\mathsf{p}_{2\to 3}^{\pi_s}$, $\mathsf{p}_{4\to 3}^{\pi_s}$ $\mathsf{b}_1^{\pi_s}, \mathsf{b}_2^{\pi_s}, \mathsf{b}_4^{\pi_s}$	$\mathsf{p}_{1\to 4}^{\pi_s}$, $\mathsf{p}_{2\to 4}^{\pi_s(0,1)}$, $\mathsf{p}_{3\to 4}^{\pi_s(0,1)}$ $\mathsf{b}_1^{\pi_s}, \mathsf{b}_2^{\pi_s}, \mathsf{b}_3^{\pi_s}$	$\mathsf{p}_{2\to 1}^{\pi_s(1,0)}$, $\mathsf{p}_{3\to 1}^{\pi_s(1,0)}$, $\mathsf{p}_{4\to 1}^{\pi_s}$ $\mathsf{b}_2^{\pi_s}, \mathsf{b}_3^{\pi_s}, \mathsf{b}_4^{\pi_s}$	$\mathsf{p}_{1\to 2}^{\pi_s}$, $\mathsf{p}_{3\to 2}^{\pi_s}$, $\mathsf{p}_{4\to 2}^{\pi_s}$ $\mathsf{b}_1^{\pi_s}, \mathsf{b}_3^{\pi_s}, \mathsf{b}_4^{\pi_s}$	$\mathsf{p}_{1\to 3}^{\pi_s}$, $\mathsf{p}_{2\to 3}^{\pi_s}$, $\mathsf{p}_{4\to 3}^{\pi_s}$ $\mathsf{b}_1^{\pi_s}, \mathsf{b}_2^{\pi_s}, \mathsf{b}_4^{\pi_s}$	$\mathsf{p}_{1\to 4}^{\pi_s}$, $\mathsf{p}_{2\to 4}^{\pi_s(1,0)}$, $\mathsf{p}_{3\to 4}^{\pi_s(1,0)}$ $\mathsf{b}_1^{\pi_s}, \mathsf{b}_2^{\pi_s}, \mathsf{b}_3^{\pi_s}$
π_a	$\mathsf{b}_2^{\pi_a}, \mathsf{b}_3^{\pi_a}$	$\mathsf{b}_1^{\pi_a}, \mathsf{b}_3^{\pi_a}$	$\mathsf{b}_1^{\pi_a}, \mathsf{b}_2^{\pi_a}$	–	$\mathsf{b}_2^{\pi_a}, \mathsf{b}_3^{\pi_a}$	$\mathsf{b}_1^{\pi_a}, \mathsf{b}_3^{\pi_a}$	$\mathsf{b}_1^{\pi_a}, \mathsf{b}_2^{\pi_a}$	–

Now we consider execution $\pi(1,1)$ and make the following claim:

Claim 7.10 *There exists* $\{r_i^{\pi(0,1)}\}_{i\in[4]}$, $\{r_i^{\pi(1,0)}\}_{i\in[4]}$ *and* $\{r_i^{\pi(1,1)}\}_{i\in[4]}$ *such that:*

(a) $\mathsf{T}_{2\leftrightarrow 3}^{\pi(0,1)} = \mathsf{T}_{2\leftrightarrow 3}^{\pi(1,0)} = \mathsf{T}_{2\leftrightarrow 3}^{\pi(1,1)}$ *and*

(b) $\mathsf{p}_{4\to i}^{\pi_s(0,1)} = \mathsf{p}_{4\to i}^{\pi_s(1,0)} = \mathsf{p}_{4\to i}^{\pi_s(1,1)}$ *and* $\mathsf{b}_4^{\pi_s(0,1)} = \mathsf{b}_4^{\pi_s(1,0)} = \mathsf{b}_4^{\pi_s(1,1)}$ *for* $i \in [3]$.

Proof: Part (a) follows from Claim 7.8. The second claim follows easily as the communication of P_4 in the synchronous round is only dependent on its random input and therefore the equalities hold when $r_4^{\pi(0,1)} = r_4^{\pi(1,0)} = r_4^{\pi(1,1)}$. □

We now describe the strategy of $\mathcal{A}$ corrupting P_1 in $\pi(1,1)$ with random inputs $\{r_i^{\pi(1,1)}\}_{i\in[4]\setminus\{1\}}$. Its private communication to P_2 is as in $\pi(1,0)$ and to P_3 is as in $\pi(0,1)$. The broadcasts messages both in synchronous and asynchronous phases are the same as the common broadcast of $\pi(0,1)$ and $\pi(1,0)$. Clearly, the view of P_2 in $\pi(1,1)$, $\mathsf{V}_2(\pi(1,1))$, with the above strategy of $\mathcal{A}$ is same as $\mathsf{V}_2(\pi(1,0))$. By correctness of π, P_2 outputs 0. On the other hand, the view of P_3 in $\pi(1,1)$, $\mathsf{V}_3(\pi(1,1))$, with the above strategy of $\mathcal{A}$ is same as $\mathsf{V}_2(\pi(0,1))$. By correctness of π, P_3 outputs 0. This violates the correctness of π as both P_2 and P_3 have input 1. Since the above breach is shown for certain set of random inputs which may be chosen with non-zero probability, the breach holds with non-zero probability too. This is a contradiction to our assumption that π securely computes f.

□

7.6 Impossibility of SVSS and SMPC with Two Synchronous Rounds

We have seen in Section 7.4 that perfectly-secure AVSS with $t < n/3$ is feasible in hybrid networks with single synchronous round. This leads to the natural question regarding the feasibility of perfectly-secure SVSS in hybrid networks with $t < n/3$. It is known that *three* synchronous rounds are necessary and sufficient for SVSS with $t < n/3$ [101]. Consequently, it is trivial to achieve SVSS with $t < n/3$ in a hybrid network with three synchronous rounds. Interestingly, it turns out that three rounds are not just sufficient, but also necessary. We prove this through the following theorem:

Theorem 7.8 *There is no perfectly-secure SVSS protocol with $n \leq 4t$ over a network that provides two synchronous rounds with broadcast oracle access prior to turning to asynchronous mode.*

Proof: Consider the setting $(n = 4, t = 1)$. The proof can easily be generalized to all $n \leq 4t$. Let $\mathcal{P} = \{P_1, P_2, P_3, P_4\}$ denote the set of parties. Without loss of generality, we assume $\mathsf{D} = P_1$. We prove the theorem by contradiction. We assume there is a perfect SVSS protocol $\pi = (\mathsf{Sh}, \mathsf{Rec})$ in the above network setting and with $(n = 4, t = 1)$. We assume that the communication done in the second synchronous round and asynchronous phase in π are done via broadcast. This holds without loss of generality since the parties can perform

point-to-point communication by exchanging random pads in the first round and then using these random pads to unmask later broadcasts [101].

Protocol Sh consist of two sub-protocols or phases: a synchronous phase and an asynchronous phase. The latter is denoted as Sh_a. The synchronous phase has two rounds. We denote the sub-protocols for the rounds as $\mathsf{Sh}_1, \mathsf{Sh}_2$. The Rec protocol is run asynchronously. So we write $\pi = (\mathsf{Sh}_1, \mathsf{Sh}_2, \mathsf{Sh}_a, \mathsf{Rec})$. $\mathsf{p}_{i\to j}^{\mathsf{Sh}_1}$ denote the transcript of the point-to-point communication done in the first synchronous round from P_i to P_j. $\mathsf{b}_i^{\mathsf{Sh}_1}$, $\mathsf{b}_i^{\mathsf{Sh}_2}$ and $\mathsf{b}_i^{\mathsf{Sh}_a}$ denote the broadcast communication done by P_i in Sh_1, Sh_2 and Sh_a respectively. We denote the view of party P_i at the end of Sh as V_i that constitutes of P_i's initial input r_i^{Sh} which includes its random coin if any (and the secret if P_i is the dealer) and the private and broadcast communication that it has received in Sh. The information sent out by P_i can be polynomially computed from the initial input and received information. Thereby, they are not considered as a part of the view. So the view of a party P_i can be defined as follows:

$$\mathsf{V}_i(\mathsf{Sh}) = \left\{r_i^{\mathsf{Sh}}, \{\mathsf{p}_{j\to i}^{\mathsf{Sh}_1}\}_{j\neq i}, \{\mathsf{b}_j^{\mathsf{Sh}_1}\}_{j\neq i}, \{\mathsf{b}_j^{\mathsf{Sh}_2}\}_{j\neq i}, \{\mathsf{b}_j^{\mathsf{Sh}_a}\}_{j\neq i}\right\}$$

In the Rec protocol, the parties simply broadcast their view from Sh protocol. Denoting $\pi(x) = (\mathsf{Sh}_1(x), \mathsf{Sh}_2(x), \mathsf{Sh}_a(x), \mathsf{Rec}(x))$ as an execution of π with the secret of the dealer P_1 as x, we now consider a couple of executions $\mathsf{Sh}(x)$ and $\mathsf{Sh}(y)$ with $x \neq y$. To differentiate the communications and views across various executions, we will parametrize the communications and views with execution names.

In $\mathsf{Sh}(x)$, P_1, P_2, P_3 and P_4 participate with initial inputs $r_1^{\mathsf{Sh}(x)}$ (defining secret x), $r_2^{\mathsf{Sh}(x)}, r_3^{\mathsf{Sh}(x)}$ and $r_4^{\mathsf{Sh}(x)}$ respectively. In $\mathsf{Sh}(y)$, P_1, P_2, P_3 and P_4 participate with initial inputs $r_1^{\mathsf{Sh}(y)}$ (defining secret y), $r_2^{\mathsf{Sh}(y)}, r_3^{\mathsf{Sh}(y)}$ and $r_4^{\mathsf{Sh}(y)}$ respectively. Assuming that $r_3^{\mathsf{Sh}(x)} = r_3^{\mathsf{Sh}(y)}$ (which can happen with non-negligible probability), we now claim that there exists $r_1^{\mathsf{Sh}(y)}, r_2^{\mathsf{Sh}(y)}, r_4^{\mathsf{Sh}(y)}$ so that $\mathsf{V}_3(\mathsf{Sh}(x)) = \mathsf{V}_3(\mathsf{Sh}(y))$. If it is not true then a corrupt P_3 can conclude that the secrets of the dealer in $\mathsf{Sh}(x)$ and $\mathsf{Sh}(y)$ are distinct which violates the perfect secrecy of π. Specifically, the equalities that follow from the equality $\mathsf{V}_3(\mathsf{Sh}(x)) = \mathsf{V}_3(\mathsf{Sh}(y))$ are as follows. When the equalities hold for two communications in $\mathsf{Sh}(x)$ and $\mathsf{Sh}(y)$, we will skip the annotation of x and y and use a common notation. So

$$\begin{aligned}
&r_3^{\mathsf{Sh}(x)} = r_3^{\mathsf{Sh}(y)} = r_3^{\mathsf{Sh}} \\
&\mathsf{p}_{j\to 3}^{\mathsf{Sh}_1(x)} = \mathsf{p}_{j\to 3}^{\mathsf{Sh}_1(y)} = \mathsf{p}_{j\to 3}^{\mathsf{Sh}_1} \text{ for } j \neq 3 \\
&\mathsf{p}_{3\to j}^{\mathsf{Sh}_1(x)} = \mathsf{p}_{3\to j}^{\mathsf{Sh}_1(y)} = \mathsf{p}_{3\to j}^{\mathsf{Sh}_1} \text{ for } j \neq 3 \\
&\mathsf{b}_j^{\mathsf{Sh}_r(x)} = \mathsf{b}_j^{\mathsf{Sh}_r(y)} = \mathsf{b}_j^{\mathsf{Sh}_r} \text{ for all } j \text{ and } r \in \{1, 2, a\}
\end{aligned} \tag{7.1}$$

For clarity, we present the views honest P_1, P_2, P_3 and P_4 in $\mathsf{Sh}(x)$ and $\mathsf{Sh}(y)$ in Table 7.3.

Table 7.3: Views of P_1, P_2, P_3, P_4 in executions $\mathsf{Sh}(x)$ and $\mathsf{Sh}(y)$

	Views in $\mathsf{Sh}(x)$				Views in $\mathsf{Sh}(y)$			
	$\mathsf{V}_1(\mathsf{Sh}(x))$	$\mathsf{V}_2(\mathsf{Sh}(x))$	$\mathsf{V}_3(\mathsf{Sh})$	$\mathsf{V}_4(\mathsf{Sh}(x))$	$\mathsf{V}_1(\mathsf{Sh}(y))$	$\mathsf{V}_2(\mathsf{Sh}(y))$	$\mathsf{V}_3(\mathsf{Sh})$	$\mathsf{V}_4(\mathsf{Sh}(y))$
Initial Input	$r_1^{\mathsf{Sh}(x)}$	$r_2^{\mathsf{Sh}(x)}$	r_3^{Sh}	$r_4^{\mathsf{Sh}(x)}$	$r_1^{\mathsf{Sh}(y)}$	$r_2^{\mathsf{Sh}(y)}$	r_3^{Sh}	$r_4^{\mathsf{Sh}(y)}$
Sh_1	$\mathsf{p}_{2\to1}^{\mathsf{Sh}_1(x)}$, $\mathsf{p}_{3\to1}^{\mathsf{Sh}_1}$, $\mathsf{p}_{4\to1}^{\mathsf{Sh}_1(x)}$ $\mathsf{b}_2^{\mathsf{Sh}_1}, \mathsf{b}_3^{\mathsf{Sh}_1}, \mathsf{b}_4^{\mathsf{Sh}_1}$	$\mathsf{p}_{1\to2}^{\mathsf{Sh}_1(x)}$, $\mathsf{p}_{3\to2}^{\mathsf{Sh}_1}$, $\mathsf{p}_{4\to2}^{\mathsf{Sh}_1(x)}$ $\mathsf{b}_1^{\mathsf{Sh}_1}, \mathsf{b}_3^{\mathsf{Sh}_1}, \mathsf{b}_4^{\mathsf{Sh}_1}$	$\mathsf{p}_{1\to3}^{\mathsf{Sh}_1}$, $\mathsf{p}_{2\to3}^{\mathsf{Sh}_1}$, $\mathsf{p}_{4\to3}^{\mathsf{Sh}_1}$ $\mathsf{b}_1^{\mathsf{Sh}_1}, \mathsf{b}_2^{\mathsf{Sh}_1}, \mathsf{b}_4^{\mathsf{Sh}_1}$	$\mathsf{p}_{1\to4}^{\mathsf{Sh}_1(x)}$, $\mathsf{p}_{2\to4}^{\mathsf{Sh}_1(x)}$, $\mathsf{p}_{3\to4}^{\mathsf{Sh}_1}$ $\mathsf{b}_1^{\mathsf{Sh}_1}, \mathsf{b}_2^{\mathsf{Sh}_1}, \mathsf{b}_3^{\mathsf{Sh}_1}$	$\mathsf{p}_{2\to1}^{\mathsf{Sh}_1(y)}$, $\mathsf{p}_{3\to1}^{\mathsf{Sh}_1}$, $\mathsf{p}_{4\to1}^{\mathsf{Sh}_1(y)}$ $\mathsf{b}_2^{\mathsf{Sh}_1}, \mathsf{b}_3^{\mathsf{Sh}_1}, \mathsf{b}_4^{\mathsf{Sh}_1}$	$\mathsf{p}_{1\to2}^{\mathsf{Sh}_1(y)}$, $\mathsf{p}_{3\to2}^{\mathsf{Sh}_1}$, $\mathsf{p}_{4\to2}^{\mathsf{Sh}_1(y)}$ $\mathsf{b}_1^{\mathsf{Sh}_1}, \mathsf{b}_3^{\mathsf{Sh}_1}, \mathsf{b}_4^{\mathsf{Sh}_1}$	$\mathsf{p}_{1\to3}^{\mathsf{Sh}_1}$, $\mathsf{p}_{2\to3}^{\mathsf{Sh}_1}$, $\mathsf{p}_{4\to3}^{\mathsf{Sh}_1}$ $\mathsf{b}_1^{\mathsf{Sh}_1}, \mathsf{b}_2^{\mathsf{Sh}_1}, \mathsf{b}_4^{\mathsf{Sh}_1}$	$\mathsf{p}_{1\to4}^{\mathsf{Sh}_1(y)}$, $\mathsf{p}_{2\to4}^{\mathsf{Sh}_1(y)}$, $\mathsf{p}_{3\to4}^{\mathsf{Sh}_1}$ $\mathsf{b}_1^{\mathsf{Sh}_1}, \mathsf{b}_2^{\mathsf{Sh}_1}, \mathsf{b}_3^{\mathsf{Sh}_1}$
Sh_2	$\mathsf{b}_2^{\mathsf{Sh}_2}, \mathsf{b}_3^{\mathsf{Sh}_2}, \mathsf{b}_4^{\mathsf{Sh}_2}$	$\mathsf{b}_1^{\mathsf{Sh}_2}, \mathsf{b}_3^{\mathsf{Sh}_2}, \mathsf{b}_4^{\mathsf{Sh}_2}$	$\mathsf{b}_1^{\mathsf{Sh}_2}, \mathsf{b}_2^{\mathsf{Sh}_2}, \mathsf{b}_4^{\mathsf{Sh}_2}$	$\mathsf{b}_1^{\mathsf{Sh}_2}, \mathsf{b}_2^{\mathsf{Sh}_2}, \mathsf{b}_3^{\mathsf{Sh}_2}$	$\mathsf{b}_2^{\mathsf{Sh}_2}, \mathsf{b}_3^{\mathsf{Sh}_2}, \mathsf{b}_4^{\mathsf{Sh}_2}$	$\mathsf{b}_1^{\mathsf{Sh}_2}, \mathsf{b}_3^{\mathsf{Sh}_2}, \mathsf{b}_4^{\mathsf{Sh}_2}$	$\mathsf{b}_1^{\mathsf{Sh}_2}, \mathsf{b}_2^{\mathsf{Sh}_2}, \mathsf{b}_4^{\mathsf{Sh}_2}$	$\mathsf{b}_1^{\mathsf{Sh}_2}, \mathsf{b}_2^{\mathsf{Sh}_2}, \mathsf{b}_3^{\mathsf{Sh}_2}$
Sh_a	$\mathsf{b}_2^{\mathsf{Sh}_a}, \mathsf{b}_3^{\mathsf{Sh}_a}, \mathsf{b}_4^{\mathsf{Sh}_a}$	$\mathsf{b}_1^{\mathsf{Sh}_a}, \mathsf{b}_3^{\mathsf{Sh}_a}, \mathsf{b}_4^{\mathsf{Sh}_a}$	$\mathsf{b}_1^{\mathsf{Sh}_a}, \mathsf{b}_2^{\mathsf{Sh}_a}, \mathsf{b}_4^{\mathsf{Sh}_a}$	$\mathsf{b}_1^{\mathsf{Sh}_a}, \mathsf{b}_2^{\mathsf{Sh}_a}, \mathsf{b}_3^{\mathsf{Sh}_a}$	$\mathsf{b}_2^{\mathsf{Sh}_a}, \mathsf{b}_3^{\mathsf{Sh}_a}, \mathsf{b}_4^{\mathsf{Sh}_a}$	$\mathsf{b}_1^{\mathsf{Sh}_a}, \mathsf{b}_3^{\mathsf{Sh}_a}, \mathsf{b}_4^{\mathsf{Sh}_a}$	$\mathsf{b}_1^{\mathsf{Sh}_a}, \mathsf{b}_2^{\mathsf{Sh}_a}, \mathsf{b}_4^{\mathsf{Sh}_a}$	$\mathsf{b}_1^{\mathsf{Sh}_a}, \mathsf{b}_2^{\mathsf{Sh}_a}, \mathsf{b}_3^{\mathsf{Sh}_a}$

Now we consider three different corrupt executions that result in the same view of the parties at the end of Sh. In all the executions, $\mathcal{A}$ schedules the messages of the parties so that $\{P_2, P_3, P_4\}$ be the set of parties who see each others message during Sh_a and Rec.

– Execution $\mathcal{E}_1$: $\mathcal{E}_1$ is similar to $\mathsf{Sh}(x)$ where the parties P_1, P_3, P_4 are honest and start with their respective initial input of execution $\mathsf{Sh}(x)$. The adversary $\mathcal{A}$ corrupts P_2 with the following corruption strategy - P_2's private communication towards P_1 and P_3 is exactly like in $\mathsf{Sh}(x)$. Its private communication to P_4 is the same as in $\mathsf{Sh}(y)$. After the private communication is done, P_2 behaves exactly like an honest P_2 in execution $\mathcal{E}_3$.

– Execution $\mathcal{E}_2$: $\mathcal{E}_2$ is similar to $\mathsf{Sh}(y)$ where the parties P_1, P_2, P_3 are honest and start with their respective initial input of execution $\mathsf{Sh}(y)$. The adversary $\mathcal{A}$ corrupts P_4 with the following corruption strategy - P_4's private communication towards P_1 and P_3 is exactly like in $\mathsf{Sh}(y)$. Its private communication to P_2 is the same as in $\mathsf{Sh}(x)$. After the private communication is done, P_4 behaves exactly like an honest P_4 in execution $\mathcal{E}_3$.

– Execution $\mathcal{E}_3$: In $\mathcal{E}_3$ the parties P_2, P_3, P_4 are honest. P_2 and P_4 start with their respective initial input of execution $\mathsf{Sh}(y)$ and $\mathsf{Sh}(x)$ respectively. P_3 starts with its initial input that is common to executions $\mathsf{Sh}(x)$ and $\mathsf{Sh}(y)$. The adversary $\mathcal{A}$ corrupts P_1. P_1 acts exactly as in $\mathsf{Sh}(y)$ towards P_2 and P_3. However its behaviour towards P_4 is as in $\mathsf{Sh}(x)$.

Through a sequence of claims, we next prove the statement that: *The view of honest P_2 in $\mathcal{E}_2$ and $\mathcal{E}_3$ will be the same. The view of honest P_3 in $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$ will be the same. The view of honest P_4 in $\mathcal{E}_1, \mathcal{E}_3$ will be the same.*

Claim 7.11 *The broadcast of every honest P_i in Sh_1 will be the same in $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$.*

Proof: In $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$, every honest P_i's broadcast in the first synchronous round is the same as $\mathsf{b}_i^{\mathsf{Sh}_1}$. The first round broadcasts depend on initial inputs. By Equation 7.1 the synchronous round broadcast of P_i for $i \in \{1,2,3,4\}$ are the same irrespective of whether its initial input is $r_i^{\mathsf{Sh}(x)}$ or $r_i^{\mathsf{Sh}(y)}$. □

Claim 7.12 *P_1's broadcasts in Sh_2 will be the same in $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$*

Proof: A careful look implies that P_1's view until the first round of $\mathcal{E}_1$ is exactly same as in $\mathsf{Sh}(x)$ (refer to $\mathsf{V}_1(\mathsf{Sh}(x))$ in Table 7.3). Similarly, P_1's view until the first round of $\mathcal{E}_2$ is exactly same as in $\mathsf{Sh}(y)$ (refer to $\mathsf{V}_1(\mathsf{Sh}(y))$ in Table 7.3). It now follows from Equation 7.1 that the second round broadcasts done by P_1 will be same as $\mathsf{b}_1^{\mathsf{Sh}_2}$ in $\mathcal{E}_1$ and $\mathcal{E}_2$. In $\mathcal{E}_3$, a corrupt P_1 can broadcast $\mathsf{b}_1^{\mathsf{Sh}_2}$. □

Claim 7.13 *The broadcast of an honest P_3 will be the same in $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$ in both $(\mathsf{Sh}_1, \mathsf{Sh}_2)$.*

Proof: It follows directly from Claim 7.11 that an honest P_3's broadcast in Sh_1 remains the same during $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$. Also, since the view of honest P_3 until the first round in all executions remains the same as in $\mathsf{Sh}(x)$ or $\mathsf{Sh}(y)$, the second round broadcasts done by P_3 must also be the same (refer to Equation 7.1). □

Lemma 7.9 *The view of honest P_2 in $\mathcal{E}_2$ and $\mathcal{E}_3$ will be the same. The view of honest P_3 in $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$ will be the same. The view of honest P_4 in $\mathcal{E}_1, \mathcal{E}_3$ will be the same.*

Proof: The common view of honest P_2 in $\mathcal{E}_2/\mathcal{E}_3$, the common view of honest P_3 in $\mathcal{E}_1/\mathcal{E}_2/\mathcal{E}_3$ and the common view of honest P_4 in $\mathcal{E}_1/\mathcal{E}_3$ are presented in Table 7.4. Given Claim 7.11-7.13, it is easy to verify our claim. We denote the broadcasts in $\mathcal{E}_1/\mathcal{E}_2/\mathcal{E}_3$ that may be different from the broadcasts of $\mathsf{Sh}(x)$ or $\mathsf{Sh}(y)$ because of view difference using a barred notation.

Table 7.4: View of honest P_2, P_3 and P_4 in $(\mathcal{E}_2/\mathcal{E}_3)$, $(\mathcal{E}_1/\mathcal{E}_2/\mathcal{E}_3)$ and $(\mathcal{E}_1/\mathcal{E}_3)$ respectively.

	$\mathsf{V}_2(\mathsf{Sh})$	$\mathsf{V}_3(\mathsf{Sh})$	$\mathsf{V}_4(\mathsf{Sh})$
Initial Input	$r_2^{\mathsf{Sh}(y)}$	r_3^{Sh}	$r_4^{\mathsf{Sh}(x)}$
Sh_1	$\mathsf{p}_{1\to2}^{\mathsf{Sh}_1(y)}, \mathsf{p}_{3\to2}^{\mathsf{Sh}_1}, \mathsf{p}_{4\to2}^{\mathsf{Sh}_1(x)}$ $\mathsf{b}_1^{\mathsf{Sh}_1}, \mathsf{b}_3^{\mathsf{Sh}_1}, \mathsf{b}_4^{\mathsf{Sh}_1}$	$\mathsf{p}_{1\to3}^{\mathsf{Sh}_1}, \mathsf{p}_{2\to3}^{\mathsf{Sh}_1}, \mathsf{p}_{4\to3}^{\mathsf{Sh}_1}$ $\mathsf{b}_1^{\mathsf{Sh}_1}, \mathsf{b}_2^{\mathsf{Sh}_1}, \mathsf{b}_4^{\mathsf{Sh}_1}$	$\mathsf{p}_{1\to4}^{\mathsf{Sh}_1(x)}, \mathsf{p}_{2\to4}^{\mathsf{Sh}_1(y)}, \mathsf{p}_{3\to4}^{\mathsf{Sh}_1}$ $\mathsf{b}_1^{\mathsf{Sh}_1}, \mathsf{b}_2^{\mathsf{Sh}_1}, \mathsf{b}_4^{\mathsf{Sh}_1}$
Sh_2	$\mathsf{b}_1^{\mathsf{Sh}_2}, \mathsf{b}_3^{\mathsf{Sh}_2}, \bar{\mathsf{b}}_4^{\mathsf{Sh}_2}$	$\mathsf{b}_1^{\mathsf{Sh}_2}, \bar{\mathsf{b}}_2^{\mathsf{Sh}_2}, \bar{\mathsf{b}}_4^{\mathsf{Sh}_2}$	$\mathsf{b}_1^{\mathsf{Sh}_2}, \bar{\mathsf{b}}_2^{\mathsf{Sh}_2}, \mathsf{b}_3^{\mathsf{Sh}_2}$
Sh_a	$\bar{\mathsf{b}}_3^{\mathsf{Sh}_a}, \bar{\mathsf{b}}_4^{\mathsf{Sh}_a}$	$\bar{\mathsf{b}}_2^{\mathsf{Sh}_a}, \bar{\mathsf{b}}_4^{\mathsf{Sh}_a}$	$\bar{\mathsf{b}}_2^{\mathsf{Sh}_a}, \bar{\mathsf{b}}_3^{\mathsf{Sh}_a}$

□

Since π is assumed to be a SVSS, the honest parties (P_2, P_3, P_4) in $\mathcal{E}_3$ must terminate after Sh without the corrupt dealer's participation in Sh_a. Otherwise, a corrupt dealer may not participate in Sh_a at all leading to an endless waiting. Now as the views of the honest parties, namely of (P_3, P_4) in $\mathcal{E}_1$ and of (P_2, P_3) in $\mathcal{E}_2$ are identical in all the three executions, they will terminate $\mathcal{E}_1$ and $\mathcal{E}_2$ after Sh without D's participation.

Now we discuss the adversary's strategy in Rec of $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$. In $\mathcal{E}_3$, $\mathcal{A}$ corrupting P_1 simply does not participate in Rec so that only $\{P_2, P_3, P_4\}$ see the views of each other from Sh. In $\mathcal{E}_1$ and $\mathcal{E}_2$, $\mathcal{A}$ corrupting P_2 and P_4 respectively presents the view of an honest P_2 and P_4 as shown in Table 7.4. $\mathcal{A}$ further delays the communication of honest P_1 so that only $\{P_2, P_3, P_4\}$ see views of each other from Sh. The above adversarial strategies in the three executions ensures that $\{P_2, P_3, P_4\}$ end up with the same view in Rec across all the three executions. Now following the same argument as presented above for the termination of Sh, $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$ must also terminate without P_1's participation in Rec protocol.

Since D is honest in $\mathcal{E}_1$ and $\mathcal{E}_2$, the parties should reconstruct x and y respectively in the Rec protocol of the executions. This follows from the correctness property of π. Now recall that the views of the parties in Rec protocol of all the three executions are the same. So in $\mathcal{E}_3$ where D is corrupt, the parties may either reconstruct x or y. This clearly violates the commitment property of π that demands there must exist a unique committed secret at the end of Sh which will be reconstructed in Rec irrespective of the behaviour of $\mathcal{A}$. Therefore we arrive at a contradiction that π is not a perfect SVSS protocol. Since the above breach is shown for certain set of initial inputs which may be chosen with non-zero probability, the breach holds with non-zero probability too. This is enough for our proof as we are concerned about impossibility of perfect SVSS. This completes the proof.

□

Since SVSS is a special case of SMPC protocol, the above theorem directly implies the impossibility of perfectly-secure SMPC protocol with $n \leq 4t$ over a hybrid network with two synchronous rounds. Thus, we get the following theorem.

Theorem 7.9 *There is no perfectly-secure SMPC protocol with $n \leq 4t$ over a network that provides two synchronous rounds with broadcast oracle access prior to turning to asynchronous mode.*

In the following section, we show that three synchronous rounds are not just necessary, but also sufficient to design an SMPC protocol with $t < n/3$ over hybrid network.

7.7 SMPC with Three Synchronous Rounds

In this section, we describe an SMPC protocol with three synchronous rounds in hybrid network and $n \geq 3t+1$, tweaking the framework of [62] and plugging in the existing 3-round VSS protocol offering t-sharing of the underlying secret [139] into the framework.

Let $F : \mathbb{F}^n \rightarrow \mathbb{F}$ be the publicly known function over field $\mathbb{F}$ where each party P_i has the input $x_i \in \mathbb{F}$ for the function and all the n parties receive the function output $F(x_1, \ldots, x_n)$. The function F is represented by an arithmetic circuit C over $\mathbb{F}$, consisting of input gates, linear gates, multiplication gates, random gates and output gates of bounded fan-in. Without loss of generality, we assume that the multiplication gates have fan-in two and the random gates have fan-in zero. Let c_I, c_L, c_M, c_R and c_O denote the number of input, linear, multiplication, random and output gates respectively in the circuit representing F. We assume $c_I = n$ and $c_O = 1$ for simplicity. We follow the standard technique of circuit evaluation where the parties "securely" evaluate each gate in the circuit in a shared/distributed fashion. The parties interact to maintain the following *invariant* for each gate in the circuit: *given t-shared inputs of the gate, say $[a]$ and $[b]$, the gate output is computed and made available among the parties in a t-shared fashion at the end of the gate evaluation.* Finally the shared circuit output is publicly reconstructed. Since each intermediate value remains secret shared, privacy follows. Due to the linearity of the sharing schemes, addition/linear gates can be locally evaluated by the parties. However, computing a multiplication (non-linear) gate following the invariant requires interaction among the parties.

A typical SMPC in information-theoretic setting consists of three phases: (a) The input phase, where the parties t-share the inputs for computation of function. (b) The preprocessing phase, where the parties jointly create t-sharing of a Beaver triple per multiplication gate [18] i.e. t-sharing of a *private random multiplication triple*. (c) The computation phase, where the shared circuit evaluation is performed using the t-shared Beaver triplets to evaluate the multiplication gates while the additional gates need only local computation. Noting that the reconstruction of t-shared secrets via oec (Section 7.2.3.2) works in the asynchronous phase with $n \geq 3t+1$, our goal is to pack the most of the protocol in the first three rounds so that the asynchronous computation involves only reconstruction of secrets and local computation. The input phase can be completed in the synchronous phase via invoking an VSS instance for every party's input. The computation phase, given the preprocessed t-shared Beaver triples, involves only reconstructions of t-shared values (two for evaluating a multiplication gate and one for evaluating an output gate) and local computation, making it feasible to run asynchronously. The primary challenge, therefore, lies in generating the t-shared Beaver triples with the help

of initial three rounds with $n \geq 3t+1$ in the preprocessing phase. Both the preprocessing and the input phase are run in parallel to take advantage of the synchronous rounds. While the input phase terminates in the synchronous phase itself, the preprocessing phase spills over to the asynchronous phase where only reconstructions and local computation are performed. We now describe the three phases below.

7.7.1 Input Phase

The protocol Input is given in Figure 7.6.

Protocol Input()

- **Input of each** $P_i \in \mathcal{P}$**:** Input $x_i \in \mathbb{F}$ for computation of circuit C.
- **Primitives Used:** Protocol Sh corresponding to a VSS protocol (Sh, Rec) of [139].
- **The Protocol:** It uses the synchronous phase with three rounds of the hybrid network. Each P_i invokes an instance of Sh as a dealer with secret as input x_i and participates in other Sh instances.

Figure 7.6: Protocol for the input phase of MPC.

Lemma 7.10 *For every possible $\mathcal{A}$ and for every possible scheduler, protocol* Input *achieves:*

- **Termination:** *All the honest parties terminate the protocol.*
- **Correctness:** *Each honest party will output its shares corresponding to t-sharing of the inputs of the parties in $\mathcal{C}$.*
- **Privacy:** *The information obtained by $\mathcal{A}$ in the protocol is distributed independently of the inputs of the honest parties in the set $\mathcal{C}$.*

Proof: The termination property is easy to verify. Correctness follows from the correctness of Sh. Privacy of Sh ensures that the information obtained by $\mathcal{A}$ during instances of Sh is independent of the private value x_i for all honest P_i. □

7.7.2 Preprocessing Phase

In this phase, the parties create raw material for evaluating the multiplication and random gates. Namely, the goal is to create $(c_M + c_R)$ t-shared random private multiplication triples. We discuss our idea for a single t-shared random private multiplication (Beaver) triple generation. We tweak the techniques of [62] used for building the preprocessing phase of an efficient MPC

with $n \geq 4t + 1$ over hybrid network with one synchronous round. The framework proposes a *share-and-extract* paradigm that has two clear steps: each party is asked to verifiably t-*share* random multiplication triples and then t-shared random multiplication triples unknown to the adversary are *extracted* from the pool of triples generated by all the parties. It is the verifiability of a multiplication triple i.e. whether $c = ab$ for a shared (a, b, c) that we simplify leveraging the three synchronous rounds and is different from [62]. We mention the difference in the relevant section. Both the tasks of verifiable sharing of multiplication triple by a party and the subsequent extraction rely on a protocol termed as *triple-transformation* protocol. In the following, we discuss the triple transformation protocol and the extraction protocol first and then conclude with the (verifiable) sharing of multiplication triples and preprocessing phase protocol.

7.7.2.1 Triple Transformation protocol

The protocol tripleTrans takes t-sharing of a set of n independent triples as input and transforms them into t-sharing of a set of n correlated triples. Namely, all the first components of the transformed triples lie on a t-degree polynomial. The same holds for the second components. The third components lie on a $2t$ degree polynomial. Furthermore, the degree $2t$ polynomial will be the product of the degree t polynomials if and only if all the input triples are multiplication triples. The protocol steps are outlined in Figure 7.7 and its properties are presented in Lemma 7.11.

Protocol tripleTrans()

- **Input:** Sharing of set of n independent triples $\{([x^{(i)}], [y^{(i)}], [z^{(i)}])\}_{i\in[n]}$
- **Primitives Used:** Protocol Beaver (Section 7.2.4)
- **Output:** Sharing of set of n correlated triples $\{([\mathsf{x}^{(i)}], [\mathsf{y}^{(i)}], [\mathsf{z}^{(i)}])\}_{i\in[n]}$
- **The Protocol:** It runs asynchronously.

1. For each $i \in [t + 1]$, the parties locally set $[\mathsf{x}^{(i)}] = [x^{(i)}]$, $[\mathsf{y}^{(i)}] = [y^{(i)}]$ and $[\mathsf{z}^{(i)}] = [z^{(i)}]$.
2. Let the points $\{(\alpha_i, \mathsf{x}^{(i)})\}_{i\in[t+1]}$ and the points $\{(\alpha_i, \mathsf{y}^{(i)})\}_{i\in[t+1]}$ define the polynomial $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$ respectively of degree at most t. The parties locally compute $[\mathsf{x}^{(i)}] = [\mathsf{X}(\alpha_i)]$ and $[\mathsf{y}^{(i)}] = [\mathsf{Y}(\alpha_i)]$, for each[a] $i \in [t + 2, n]$.
3. For $i \in [t + 2, n]$, the parties invoke Beaver() on $([\mathsf{x}^{(i)}], [\mathsf{y}^{(i)}])$ and $([x^{(i)}], [y^{(i)}], [z^{(i)}])$ to compute $[\mathsf{z}^{(i)}]$.

4. Let the points $\{(\alpha_i, \mathsf{z}^{(i)})\}_{i\in[n]}$ define the polynomial $\mathsf{Z}(\cdot)$ of degree at most $2t$. The parties output $\{([\mathsf{x}^{(i)}], [\mathsf{y}^{(i)}], [\mathsf{z}^{(i)}])\}_{i\in[n]}$ and terminate.

[a] Computing a new point on a polynomial of degree t is a linear function of $t+1$ given unique points on the same polynomial.

Figure 7.7: Protocol for transforming a set of independent shared triples to a set of correlated shared triples.

The protocol clearly implies the following *one-to-one correspondence* between the input and the output triples, in an *error-free* fashion: **(a)** The i^{th} output triple is a *multiplication* triple if and only if the i^{th} input triple is a multiplication triple and **(b)** The i^{th} output triple will be known to $\mathcal{A}$ if and only if the i^{th} input triple is known to $\mathcal{A}$. The first property guarantees that the relation $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ is true if and only if all the n input triples are multiplication triples. On the other hand, the second property guarantees that if $\mathcal{A}$ knows t' input triples, then it implies $t+1-t'$ "degree of freedom" in the polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$, provided $t' \leq t$; if $t' > t$, then $\mathcal{A}$ will know all the output triples. So we have the following lemma.

Lemma 7.11 *([62]) Let $\{([x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)\}_{i\in[n]}$ be a set of n t-shared triples. Then for every possible $\mathcal{A}$, protocol* tripleTrans *achieves:*

- **Termination**: *All the honest parties eventually terminate the protocol*
- **Correctness**: *There exist polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$ of degree t, t and $2t$ respectively, such that $\mathsf{X}(\alpha_i) = \mathsf{x}^{(i)}, \mathsf{Y}(\alpha_i) = \mathsf{y}^{(i)}$ and $\mathsf{Z}(\alpha_i) = \mathsf{z}^{(i)}$ holds for $i \in [n]$ and $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ holds if and only if all the input triples are multiplication triples.*
- **Privacy**: *If $\mathcal{A}$ knows $t' < t$ input triples then $\mathcal{A}$ learns t' values on $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$.*

The protocol slightly differs from that of [62] in terms of the degrees used. Namely, their polynomials $\mathsf{X}(), \mathsf{Y}(), \mathsf{Z}()$ are of degree $\frac{3t}{2}, \frac{3t}{2}$ and $3t$ respectively to facilitate generating one multiplication triple at the expense of linear (in n) communication cost, leveraging presence of $n \geq 4t+1$ parties. Looking ahead, our verifiable triple sharing with $n \geq 3t+1$ does not work for the above degrees and the degrees we use is the only possibility that works.

7.7.2.2 Triple Extraction protocol

This protocol 'extracts' sharing of one multiplication triple from sharing of a set of n correlated multiplication triples where $n \geq 3t+1$. If $\mathcal{A}$ knows at most t of the correlated triples, then the extracted triple is random and unknown to $\mathcal{A}$ subject to the fact that it is a multiplication triple. The protocol is outlined in Figure 7.8 and its properties stated and proved in Lemma 7.12.

Protocol tripleExt()

- **Input:** Correlated shared multiplication triples $\{([\mathsf{x}^{(i)}], [\mathsf{y}^{(i)}], [\mathsf{z}^{(i)}])\}_{i\in[n]}$ with polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$ of degree t, t and $2t$ respectively, such that $\mathsf{X}(\alpha_i) = \mathsf{x}^{(i)}, \mathsf{Y}(\alpha_i) = \mathsf{y}^{(i)}$ and $\mathsf{Z}(\alpha_i) = \mathsf{z}^{(i)}$ for $i \in [n]$ and $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$.
- **Output:** A shared multiplication triple $([a], [b], [c])$
- **The Protocol:** It runs asynchronously. The parties locally compute $[a] = [\mathsf{X}(\beta)]$, $[b] = [\mathsf{Y}(\beta)]$ and $[c] = [\mathsf{Z}(\beta)]$ and terminate, where β is a publicly known point distinct from $\alpha_1, \ldots, \alpha_n$.

Figure 7.8: Protocol for extracting sharing of a multiplication triple from sharing of a set of n multiplication triples, where $n \geq 3t + 1$.

Lemma 7.12 *([62]) Given a set of correlated shared multiplication triples* $\{([\mathsf{x}^{(i)}], [\mathsf{y}^{(i)}], [\mathsf{z}^{(i)}])\}_{i\in[n]}$ *with polynomials* $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ *and* $\mathsf{Z}(\cdot)$ *of degree* t, t *and* $2t$ *respectively, such that* $\mathsf{X}(\alpha_i) = \mathsf{x}^{(i)}, \mathsf{Y}(\alpha_i) = \mathsf{y}^{(i)}$ *and* $\mathsf{Z}(\alpha_i) = \mathsf{z}^{(i)}$ *for* $i \in [n]$ *and* $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$*, for every possible* $\mathcal{A}$ *protocol* tripleExt *achieves:*

- **Correctness:** *The output triple* $([a], [b], [c])$ *is a multiplication triple.*
- **Privacy:** *If* $\mathcal{A}$ *knows at most* t *input triples, then the view of* $\mathcal{A}$ *is distributed independently of the output multiplication triple* $([a], [b], [c])$.

7.7.2.3 Verifiable Multiplication Triple Sharing Protocol

To keep the presentation simple, we show how a specific party, say D shares a single multiplication triple verifiably. D first t-shares n independent multiplication triples using $3n$ instances of Sh (one for each component of the triple) of [139]. The triple transformation protocol is invoked to transform his set of n t-shared independent triples to a set of n t-shared correlated triples with the underlying polynomials as $\mathsf{X}(\cdot), \mathsf{Y}(\cdot), \mathsf{Z}(\cdot)$ of degree t, t and $2t$ respectively. Now the input triples are verified for their product relation via a pubic verification of the relation $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$, in an error-free manner, where each party P_i supervises the public verification of $\mathsf{Z}(\alpha_i) = \mathsf{X}(\alpha_i)\mathsf{Y}(\alpha_i)$. An honest supervisor will make sure the relation holds without any error. Now since there are at least $2t + 1$ honest supervisors and $\mathsf{Z}(\cdot)$ is a polynomial of degree at most $2t$, we can conclude that $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$. The verification led by party P_i is conducted as follows:

P_i t-shares a dummy random multiplication triple $([f_i], [g_i], [h_i])$ using 3 instances of Sh in the synchronous phase to help compute $[\mathsf{X}(\alpha_i)\mathsf{Y}(\alpha_i]$ via Beaver's technique. Then the difference of $\mathsf{Z}(\alpha_i)$ and $\mathsf{X}(\alpha_i)\mathsf{Y}(\alpha_i)$ is reconstructed via oec. When the difference is zero, it can be concluded

that $\mathsf{Z}(\alpha_i) = \mathsf{X}(\alpha_i)\mathsf{Y}(\alpha_i)$. Else, the triple $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ is reconstructed using oec without compromising privacy since the corrupt party that is either D or P_i already knows the triple. If it is found to be a non-multiplication triple, D can be identified as corrupt and default t-shared multiplication triple is considered corresponding to D.

When $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ is verified, the t-sharing of $(\mathsf{X}(\beta), \mathsf{Y}(\beta), \mathsf{Z}(\beta))$ for a public value β is taken as the multiplication triple dealt by D. We note that the above public verification of $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ will not work when the degree of the polynomials are $\frac{3t}{2}, \frac{3t}{2}, 3t$ respectively (as in [62]). With these degrees, $\mathsf{Z}(\alpha_i) = \mathsf{X}(\alpha_i)\mathsf{Y}(\alpha_i)$ needs to be verified by at least $3t+1$ *honest* parties, whereas we have just $2t+1$ honest parties in the population. On the other hand, the polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ should be of degree at least t, otherwise an adversary corrupting t parties would get access to t points of each of these polynomials (via Beaver's technique) learning $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ entirely. Thus, the only feasible choice for the degree of the polynomials are t, t, $2t$ respectively.

We differ from [62] in the following aspect. Using three synchronous rounds, we get t-shared dummy triples. Whereas, the dummy triples [62] are not in t-shared form due to the availability of a single synchronous round. Consequently, our protocol only needs to deal with malicious behavior of sharing a non-multiplication dummy triple, while [62] needs to handle additionally the case when the dummy triple is not t-shared. Working with $n \geq 4t+1$, [62] leverages the larger honest population to handle this issue. We now present the protocol in Figure 7.9 and prove its properties in Lemma 7.13.

Lemma 7.13 *Protocol* tripleSh() *achieves the following for every possible* $\mathcal{A}$*:*

- **Termination:** *All honest parties terminate the protocol corresponding to every* D*.*
- **Correctness:** *The output triple* $([p], [q], [r])$ *is a multiplication triple.*
- **Privacy:** *If* D *is honest, then the view of* $\mathcal{A}$ *is distributed independently of the output multiplication triple* $([p], [q], [r])$*.*

Proof: **Termination:** The termination property is easy to verify and follows directly from the termination property of synchronous phase, tripleTrans, Beaver and oec.

Correctness: We first consider the case of an honest D, where the t-shared triples $\{([x^{(k)}], [y^{(k)}], [z^{(k)}])\}_{k\in[n]}$ will be multiplication triples. By the property of tripleTrans (Lemma 7.11), $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ holds and $\{([\mathsf{x}^{(k)}], [\mathsf{y}^{(k)}], [\mathsf{z}^{(k)}])\}_{k\in[n]}$ will be multiplication triples. It now follows from the property of Beaver that $[\bar{z}^{(k)} = \mathsf{x}^{(k)}\mathsf{y}^{(k)}]$ computed by honest parties using dummy multiplication triple $([f_k], [g_k], [h_k])$ shared by honest P_k must correspond to $\mathsf{z}^{(k)}$ and lead to reconstruction

of $\gamma^{(k)} = 0$ (correctness of oec). However, dummy triples given by a corrupt party P_k may lead to $\gamma^{(k)} \neq 0$ being reconstructed. However, the verification would be still be successful since the triple $\{([\mathsf{x}^{(k)}], [\mathsf{y}^{(k)}], [\mathsf{z}^{(k)}])\}$ would be reconstructed and found to be a multiplication triple (correctness of oec). Finally, the correctness of $\mathsf{tripleExt}$ (Lemma 7.12) ensures that the output triple $([p], [q], [r])$ is a multiplication triple.

In case of corrupt D, the output may be either a default t-shared multiplication triple or the triple computed as per the protocol. In the former case, correctness holds trivially. We now consider the latter case. Note that it suffices to prove that corresponding to each honest P_k, $\{([\mathsf{x}^{(k)}], [\mathsf{y}^{(k)}], [\mathsf{z}^{(k)}])\}$ is a multiplication triple, which will confirm that $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ holds (since $\mathsf{Z}(\cdot)$ is a $2t$ degree polynomial uniquely determined by $2t+1$ points). This would imply that the triples shared by D are multiplication triples by property of $\mathsf{tripleTrans}$ (Lemma 7.11); the proof would now follow from previous argument. We observe that the dummy triple shared by honest P_k must be a multiplication triple. Therefore, it follows from the correctness of Beaver that all honest parties compute $[\bar{z}^{(k)} = \mathsf{x}^{(k)}\mathsf{y}^{(k)}]$. Clearly, if $\{([\mathsf{x}^{(k)}], [\mathsf{y}^{(k)}], [\mathsf{z}^{(k)}])\}$ shared by corrupt D is not a multiplication triple, $\gamma^{(k)} = \bar{z}^{(k)} - \mathsf{z}^{(k)} \neq 0$ would be reconstructed. However, in this case the triple $\{([\mathsf{x}^{(k)}], [\mathsf{y}^{(k)}], [\mathsf{z}^{(k)}])\}$ would be reconstructed and found to be non-multiplication triple and default t-shared multiplication triple would be output which contradicts our assumption. So correctness holds for corrupt D.

Privacy: Since D is honest, all shared triples $\{([x^{(k)}], [y^{(k)}], [z^{(k)}])\}_{k\in[n]}$ will be random and unknown to $\mathcal{A}$. It thereby follows from properties of $\mathsf{tripleTrans}$ (Lemma 7.11) that all shared triples $\{([\mathsf{x}^{(k)}], [\mathsf{y}^{(k)}], [\mathsf{z}^{(k)}])\}_{k\in[n]}$ will be random and $\mathcal{A}$ will not learn any point on $\mathsf{X}(\cdot), \mathsf{Y}(\cdot), \mathsf{Z}(\cdot)$. Now there can be at most t corrupted P_is and corresponding to them, $\mathcal{A}$ will learn the multiplication triple $\{([\mathsf{x}^{(i)}], [\mathsf{y}^{(i)}], [\mathsf{z}^{(i)}])\}$ during its verification, as $\mathcal{A}$ will know the corresponding dummy triple $([f_i], [g_i], [h_i])$ used for its verification. However, corresponding to the honest P_is, the random dummy multiplication triples $([f_i], [g_i], [h_i])$ will be t-shared and will be not known to $\mathcal{A}$. This further implies that while computing $[\bar{z}^{(i)} = \mathsf{x}^{(i)}\mathsf{y}^{(i)}]$ using $([f_i], [g_i], [h_i])$, no additional information about the multiplication triple$\{([\mathsf{x}^{(i)}], [\mathsf{y}^{(i)}], [\mathsf{z}^{(i)}])\}$ will be leaked to $\mathcal{A}$. Finally, the sharings $[\bar{z}^{(i)}]$ and $[\mathsf{z}^{(i)}]$ will be independent, except that $\bar{z}^{(i)} = \mathsf{z}^{(i)}$ and so $\mathcal{A}$ will already know that $\gamma^{(i)} = 0$ and thus no new information is added to its view after the public reconstruction of $[\gamma^{(i)}]$. So overall, $\mathcal{A}$ will learn t values on $\mathsf{X}(\cdot), \mathsf{Y}(\cdot), \mathsf{Z}(\cdot)$.

□

Protocol tripleSh()

– **Output:** A verifiably t-shared multiplication triple corresponding to specific party D.

– **The Protocol:** The protocol runs over both the synchronous and asynchronous phase.

Synchronous Phase:

1. D invokes $3n$ instances of Sh for a SVSS of [139] to t-share $\{([x^{(k)}], [y^{(k)}], [z^{(k)}])\}_{k\in[n]}$.
2. Every party P_i $(i \in [n])$ invokes 3 instances of Sh for a SVSS of [139] to t-share a random multiplication triple $([f_i], [g_i], [h_i])$.
3. All parties invoke $\mathsf{tripleTrans}()$ on $\{([x^{(k)}], [y^{(k)}], [z^{(k)}])\}_{k\in[n]}$ to compute $\{([\mathsf{x}^{(k)}], [\mathsf{y}^{(k)}], [\mathsf{z}^{(k)}])\}_{k\in[n]}$.

Asynchronous Phase: Every P_i does the following

1. Invoke $\mathsf{Beaver}()$ on $\{([\mathsf{x}^{(k)}], [\mathsf{y}^{(k)}])\}$ and $\{([f_k], [g_k], [h_k])\}$ to compute $[\bar{z}^{(k)}]$ for $k = [n]$.
2. For $k = [n]$, compute locally $[\gamma^{(k)}]$ as $[\gamma^{(k)}] = [\bar{z}^{(k)}] - [\mathsf{z}^{(k)}]$ and reconstruct using oec. If $\gamma^{(k)} \neq 0$ for some k, reconstruct $\{([\mathsf{x}^{(k)}], [\mathsf{y}^{(k)}], [\mathsf{z}^{(k)}])\}$ using 3 instances of oec, one for each component of triple. If reconstructed triple is a non-multiplication triple, output a default t-shared multiplication triple. Otherwise invoke $\mathsf{tripleExt}()$ on $\{([\mathsf{x}^{(k)}], [\mathsf{y}^{(k)}], [\mathsf{z}^{(k)}])\}_{k\in[n]}$ and output the t-shared multiplication triple $([p], [q], [r])$. which is the output of $\mathsf{tripleExt}$.

Figure 7.9: Protocol for Verifiable Multiplication Triple Sharing.

7.7.2.4 The preprocessing phase protocol

For each Beaver triple (random secret multiplication triple) to be used for a multiplication gate, each of the n parties acts as D in an instance of $\mathsf{tripleSh}$ to generate a single verified t-shared multiplication triple. These n t-shared multiplication triples among which at most t may be known to the adversary are again transformed via the triple transformation protocol $\mathsf{tripleTrans}$ and a single t-shared Beaver triple is extracted via $\mathsf{tripleExt}$ protocol. The protocol is presented in Figure 7.10 and we prove its properties in Lemma 7.14.

Protocol $\mathsf{preProc}()$

– **Primitives Used:** Protocol $\mathsf{tripleSh}$, $\mathsf{tripleTrans}$ and $\mathsf{tripleExt}$.

– **The Protocol:** It uses both the synchronous and asynchronous phase. The following is repeated in parallel for $c_M + c_R$ times. Each P_i invokes an instance of $\mathsf{tripleSh}$ as a dealer to verifiably t-share a multiplication triple, say $([p^{(i)}], [q^{(i)}], [r^{(i)}])$ and participates in the instances of others. The parties execute $\mathsf{tripleTrans}(\{([p^{(i)}], [q^{(i)}], [r^{(i)}])\}_{i\in[n]})$ and output $\{([a^{(i)}], [b^{(i)}], [c^{(i)}])\}_{i\in[n]}$. Finally, $\mathsf{tripleExt}$ is executed on $\{([a^{(i)}], [b^{(i)}], [c^{(i)}])\}_{i\in[n]}$ to extract and output a single Beaver triple.

Figure 7.10: Protocol for the input phase of MPC.

Lemma 7.14 *For every possible $\mathcal{A}$, the protocol* prePoc() *achieves:*

– **Termination:** *All honest parties terminate the protocol.*

– **Correctness:** *$c_M + c_R$ multiplication triples will be t-shared.*

– **Privacy:** *The view of $\mathcal{A}$ will be independent of the output multiplication triples.*

Proof: Termination and privacy follows directly from the properties of the subprotocols tripleSh, tripleTrans and tripleExt. For correctness, consider computation of a single preprocessed triple. By property of tripleSh (Lemma 7.13), $([p^{(i)}], [q^{(i)}], [r^{(i)}])$ is guaranteed to be a multiplication triple for each $i \in [n]$ which implies that $\{([a^{(i)}], [b^{(i)}], [c^{(i)}])\}_{i\in[n]}$ is a set of multiplication triples (property of tripleTrans). Finally, correctness of tripleExt executed on $\{([a^{(i)}], [b^{(i)}], [c^{(i)}])\}_{i\in[n]}$ ensures that t-sharing of a multiplication triple is obtained.

□

7.7.3 Computation Phase

We now present our computation phase protocol computation in Figure 7.11 that securely evaluates the given circuit C on a gate by gate basis as discussed earlier. We use Beaver's circuit randomization technique to evaluate the multiplication and random gates with the help of preprocessed $c_M + c_R$ random t-shared multiplication-triples.

Protocol computation()

– **Input of the parties in $\mathcal{P}$:** $\{([a^{(i)}], [b^{(i)}], [c^{(i)}])\}_{i\in[c_M+c_R]}$ with the knowledge that $([a^{(i)}], [b^{(i)}], [c^{(i)}])$ is *associated* with the ith multiplication gate in the circuit C for $i \in [c_M]$ and $([a^{(c_M+i)}], [b^{(c_M+i)}], [c^{(c_M+i)}])$ is associated with the ith random gate in the circuit C for $i \in [c_R]$.

– **Common Inputs:** A field $\mathbb{F}$ and the circuit C.

– **The Protocol:** The protocol runs asynchronously. For all the gates in the circuit C the (honest) parties in $\mathcal{P}$ do the following (depending upon the type of gate):

1. **Addition/Linear Gate.** The parties locally apply the linear function on their respective shares of the inputs of the gate.

2. **Random Gate.** If this is the ith random gate in the circuit then the parties locally output their shares corresponding to the sharing $[a^{(c_M+i)}]$.

3. **Multiplication Gate.** For the ith multiplication gate with inputs $[x^{(i)}]$ and $[y^{(i)}]$ the parties invoke protocol Beaver() with $([a^{(i)}], [b^{(i)}], [c^{(i)}])$ and output $[x^{(i)}y^{(i)}]$.

4. **Output Gate.** Let $[s]$ be the t-sharing associated with the output gate. The parties execute an instance of oec to reconstruct s and terminate.

Figure 7.11: The computation phase protocol

The properties of the protocol computation() are stated below.

Lemma 7.15 *Given t-sharings $\{([a^{(i)}], [b^{(i)}], [c^{(i)}])\}_{i\in[c_M+c_R]}$ of $c_M + c_R$ random multiplication-triples and t-sharings of the inputs of the parties (for the computation), protocol* computation *achieves the following for every possible $\mathcal{A}$:*

– **Termination.** *All honest parties eventually terminate the protocol.*

– **Correctness.***: All the gates are evaluated correctly.*

– **Privacy.** *For every gate in the circuit, the evaluation of the gate reveals no additional information about the inputs and the output of the gate to $\mathcal{A}$.*

Proof: The **correctness** and **termination** property follows from the correctness and termination property of oec. The **privacy** follows from privacy of protocol Beaver() and the fact that the evaluation of random and linear gates require no communication among the parties. □

7.7.4 SMPC Protocol

The steps of the MPC protocol are outlined in Figure 7.12.

Protocol mpc

– **Input of $P_i \in \mathcal{P}$:** x_i .

– **Common Inputs:** A field $\mathbb{F}$ and circuit C

– **Protocol:** It assumes a synchronous phase with three rounds followed by an asynchronous phase. The parties invoke Input and preProc together. After terminating both the protocols, they invoke computation() and output the output of computation().

Figure 7.12: An Asynchronous MPC protocol.

Theorem 7.10 mpc *is an SMPC protocol over a hybrid network with three synchronous rounds.*

Proof: The termination, correctness and privacy follows from termination, correctness and privacy of Input (Lemma 7.10), preProc (Lemma 7.14) and computation (Lemma 7.15) protocols. □

7.8 Conclusion and Open Problems

In this chapter, we have examined the feasibility of VSS and MPC protocols in hybrid networks achieving properties obtainable in fully synchronous and asynchronous networks. For asynchronous protocols, we attempted to bridge the gap in the fault-tolerance with synchronous protocols by utilizing initial synchronous rounds present in hybrid networks. We proved that to achieve fault tolerance of synchronous protocols, while one synchronous round is both necessary and sufficient for perfectly-secure AVSS, the same does not hold for AMPC. The latter result implies that atleast two initial synchronous rounds are necessary for AMPC; finding corresponding tight upper bound remains an interesting open question.

Next, for synchronous protocols we explored whether number of synchronous rounds could be reduced leveraging the asynchronous phase available in hybrid networks. Interestingly, the answer turns out to be negative. We showed that three synchronous rounds known to be sufficient for SVSS is also necessary. This result implies a lower bound of three rounds for SMPC in hybrid network, corresponding to which we present a matching upper bound using existing techniques. We conclude that three synchronous rounds are sufficient to design perfectly-secure SMPC (and thus AMPC) in hybrid network.

Chapter 8

Conclusion

In this chapter, we summarize the contributions of this thesis and list some open problems for future work.

8.1 Summary of Results and Open Problems

8.1.1 MPC with Small Population

8.1.1.1 On the Exact Round Complexity of 3PC.

We settled the question of the exact round complexity of 3PC protocols with one active corruption in the *plain model* achieving a range of security notions, namely sa, ua, fn and god in a setting with pair-wise private channels and without or with a broadcast channel. In the minimal setting of pairwise-private channels, 3PC with sa is known to be feasible in just two rounds, while god is infeasible to achieve irrespective of the number of rounds. Settling the quest for exact round complexity of 3PC in this setting, we show that three rounds are necessary and sufficient for ua and fn. Extending our study to the setting with an additional broadcast channel, we show that while ua is achievable in just two rounds, three rounds are necessary and sufficient for fn and god.

Interesting Inferences: Our results gives the following insights regarding round complexity of MPC protocols. First, it implies the tightness of several known constructions. Our lower bound for fairness assuming broadcast implies that for 2-round fair (or guaranteed output delivery) protocols with one corruption, the number of parties needs to be at least four, making the 4PC protocol of [129] an optimal one. Next, the lower bound result of unanimous abort without broadcast immediately implies tightness of the 3PC protocol of [129] achieving selective abort

in two rounds, in terms of security achieved. Lastly, our results highlight that the availability of broadcast only impacts the round complexity of ua and god, leaving the round complexity of sa and fn unperturbed in the 3PC setting.

Open Problems: While our lower bound results extend for any number of parties in honest majority setting., our upper bounds do not extend for $t > 1$. We leave the question of designing round-optimal protocols for the general case with various security notions under the assumption of injective one-way functions.

8.1.1.2 Fast Secure Computation for 3PC and 4PC over the Internet.

Assuming the minimal model of pairwise-private channels, we present two protocols that involve computation and communication of a single GC– (a) a 4-round 3PC with fn, (b) a 5-round 4PC with god. Empirically, our protocols are on par with the best known 3PC protocol of [159] that only achieves sa, in terms of the computation time, LAN runtime, WAN runtime and communication cost. In fact, our 4PC outperforms the 3PC of [159] significantly in terms of per-party computation and communication cost. With an extra GC, we improve the round complexity of our 4PC to four rounds. The only 4PC in our setting, given by [129], involves 12 GCs. Assuming an additional broadcast channel (inevitable due to known impossibility), we present a 5-round 3PC with god that involves computation and communication of a single GC.

Interesting Inferences & Open Problems: Our constructions highlight that achieving strong notions of fn and god is possible with nominal overhead over abort security incase of 3PC / 4PC with single corruption. This gives promise of more efficient fn and god protocols in practice. We leave the question of designing practically-efficient protocols for the general honest majority setting as open.

8.1.2 On the Exact Round Complexity of Best-of-Both-Worlds Multi-party Computation

We nearly settle the exact round complexity of two classes of BoBW protocols differing on the security achieved in the honest-majority setting, namely god and fn respectively, under the assumption of no setup (plain model), public setup (CRS) and private setup (CRS + PKI or simply PKI). The former class necessarily requires the number of parties to be strictly more than the sum of the bounds of corruptions in the honest-majority and dishonest-majority setting, for a feasible solution to exist. Demoting the goal to the second-best attainable security in the honest-majority setting, the latter class needs no such restriction.

Assuming a network with pair-wise private channels and a broadcast channel, we show that 5 and 3 rounds are necessary and sufficient for the class of BoBW MPC with fn under the assumption of 'no setup' and 'public and private setup' respectively. For the class of BoBW MPC with god, we show necessity and sufficiency of 3 rounds for the public setup case and 2 rounds for the private setup case. In the no setup setting, we show the sufficiency of 5 rounds, while the known lower bound is 4. Our results remain unaffected when security with abort and fairness are upgraded to their identifiable counterparts.

Interesting Inferences & Open Problems: Our results demonstrate that the optimal round complexity of BoBW protocols are on a positive note at most one more, compared to the maximum of the needs of the honest-majority and dishonest-majority setting. This substantiates that the desirable features of BoBW protocols over traditional protocols can be attained without compromising on the number of rounds. If the same holds true regarding computation and communication efficiency as well, then BoBW protocols would indeed be the best-suited choice for real-life scenarios. We leave this question about exploring the other complexity measures of BoBW protocols as open and note that the work of [99] makes progress in this direction.

8.1.3 On the Round Complexity of Fair and Robust MPC against Dynamic and Boundary Adversaries

We settled the exact round complexity of fair and robust (achieving god) MPC tolerating dynamic and boundary adversaries. As it turns out, $\lceil n/2 \rceil + 1$ rounds are necessary and sufficient for fair as well as robust MPC tolerating dynamic corruption. The non-constant barrier raised by dynamic corruption can be sailed through for a boundary adversary. The round complexity of 3 and 4 is necessary and sufficient for fn and god protocols respectively, with the latter having an exception of allowing 3 round protocols in the presence of a single active corruption. While all our lower bounds assume pair-wise private and broadcast channels and are resilient to the presence of both public (CRS) and private (PKI) setup, our upper bounds are broadcast-only and assume only public setup. The traditional and popular setting of malicious-minority, being restricted compared to both dynamic and boundary setting, requires 3 and 2 rounds in the presence of public and private setup respectively for both fair as well as robust protocols.

Interesting Inferences and Open Problems: The results provide us further insights regarding how disparity in adversarial setting affects round complexity. Note that the round complexity of fair protocols in the CRS model against an adversary corrupting minority of parties maliciously,

remains unaffected in the setting of boundary adversary; which is a stronger variant of the former. On the other hand, this switch of adversarial setting causes the lower bound of robust protocols in the model assuming both CRS and PKI to jump from 2 to 4. Lastly, the gravity of dynamic corruption on round complexity is evident in the leap from constant-rounds of 3, 4 in the boundary corruption case to $\lceil n/2 \rceil + 1$. An interesting open question is to construct tight upper bounds or come up with new lower bounds in the plain model.

8.1.4 On the Power of Hybrid Networks in Multi-Party Computation

We address the following fundamental question: *What is the minimum number of initial synchronous rounds necessary and sufficient in a hybrid network to construct perfectly-secure AVSS and AMPC protocols with the same fault-tolerance of synchronous protocols?* On the positive side, we show that *one* synchronous round is sufficient for AVSS which is clearly optimal. On the negative side, we show the same is not true for AMPC. Notably *no broadcast* oracle is invoked in the synchronous round of our AVSS protocol. The latter result on AMPC implies at least *two* initial synchronous rounds are necessary for MPC. With three synchronous rounds, we design a perfectly-secure SMPC (and thus AMPC) protocol in this work. We further investigate if the asynchronous phase of the hybrid network can be leveraged to save on the synchronous rounds required for SVSS and SMPC. It is known that *three* synchronous rounds are necessary and sufficient for SVSS with $t < n/3$ [101]. This makes the feasibility of SVSS with $t < n/3$ in a hybrid network with three synchronous rounds trivial. The same question seems intriguing when one or two synchronous rounds are assumed. We answer this question in the negative.

Interesting Inferences: Our results reinforce several general beliefs in the context of hybrid networks: **(a)** AMPC is harder to achieve than AVSS , **(b)** SVSS is harder to achieve than AVSS with the same resilience, **(c)** perfectly-secure SMPC is harder to achieve than cryptographic SMPC (follows from our work and the result of [23]).

Open Problems: The question of designing an AMPC protocol in a hybrid network with two synchronous rounds with or without broadcast oracle access is left as an interesting open question.

Bibliography

[1] Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. An almost-surely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience. In *ACM Symposium on Principles of Distributed Computing, PODC*, 2008. 16

[2] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 387–404, 2014. 103

[3] Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 468–499, 2017. 5, 11, 156, 212

[4] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 395–424, 2018. 6, 11, 13, 16, 34, 156, 159, 161, 175, 176, 178, 226, 239, 253

[5] Benny Applebaum, Barak Arkis, Pavel Raykov, and Prashant Nalini Vasudevan. Conditional disclosure of secrets: Amplification, closure, amortization, lower-bounds, and separations. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 727–757, 2017. 101

[6] Toshinori Araki, Assaf Barak, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. DEMO: high-throughput secure three-party computation of kerberos ticket gen-

eration. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1841–1843, 2016. 104

[7] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 805–817, 2016. 7, 103, 104

[8] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 843–862, 2017. 104, 105

[9] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly-secure multiparty computation. *Electronic Colloquium on Computational Complexity (ECCC)*, 2011. 284

[10] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 483–501, 2012. 161, 174, 179, 180, 181, 192

[11] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 483–501, 2012. 34, 103

[12] Hagit Attiya and Jennifer L. Welch. *Distributed computing - fundamentals, simulations, and advanced topics (2. ed.)*. Wiley series on parallel and distributed computing. Wiley, 2004. 275

[13] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 590–609, 2011. 15, 33, 272

[14] Michael Backes, Fabian Bendun, Ashish Choudhury, and Aniket Kate. Asynchronous MPC with a strict honest majority using non-equivocation. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 10–19, 2014. 275

[15] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal MPC. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 459–487, 2018. xi, 5, 11, 13, 156, 159, 160, 169, 170, 211, 220, 221, 222

[16] Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Threshold multi-key fhe and applications to round-optimal mpc. Cryptology ePrint Archive, Report 2018/580, 2018. 6, 13, 16, 34, 159, 226

[17] D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*. Springer Verlag, 1991. 156

[18] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 420–432, 1991. 11, 103, 283, 284, 307

[19] Donald Beaver, Silvio Micali, and Phillip 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, 1990. 11, 24, 103, 156, 212

[20] Zuzana Beerliová-Trubíniová and Martin Hirt. Simple and efficient perfectly-secure asynchronous MPC. In *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*, pages 376–392, 2007. xii, 103, 281, 283

[21] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, pages 213–230, 2008. 103, 274

[22] Zuzana Beerliová-Trubíniová, Matthias Fitzi, Martin Hirt, Ueli M. Maurer, and Vassilis Zikas. MPC vs. SFE: perfect security in a unified corruption model. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, pages 231–250, 2008. 225

[23] Zuzana Beerliová-Trubíniová, Martin Hirt, and Jesper Buus Nielsen. On the theoretical gap between synchronous and asynchronous MPC protocols. In *ACM Symposium on Principles of Distributed Computing, PODC*, 2010. 2, 18, 274, 321

[24] Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov. $1/p$-secure multiparty computation without honest majority and the best of both worlds. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 277–296, 2011. 157, 158

[25] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 547–557, 1989. 165

[26] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, pages 134–153, 2012. 27, 28

[27] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796, 2012. 24, 26, 135, 140

[28] Aner Ben-Efraim and Eran Omri. Concrete efficiency improvements for multiparty garbling with an honest majority. In *Progress in Cryptology - LATINCRYPT 2017 - 5th International Conference on Cryptology and Information Security in Latin America, Havana, Cuba, September 20-22, 2017, Revised Selected Papers*, pages 289–308, 2017. 104

[29] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 578–590, 2016. 104

[30] Michael Ben-Or, Shafi Goldwasser, and Avi 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, 1988. 1, 6, 11, 16, 19, 103, 156, 274

[31] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 52–61, 1993. 274, 277

[32] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17, 1994*, pages 183–192, 1994. 274, 279

[33] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 663–680, 2012. 103, 274

[34] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pages 169–188, 2011. 103

[35] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 500–532, 2018. x, xv, 5, 6, 13, 15, 16, 159, 160, 161, 162, 169, 170, 178, 179, 181, 182, 183, 184, 185, 186, 188, 190, 191, 192, 204, 207, 209, 211, 223, 226, 227, 236, 237, 243, 252, 256

[36] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 103–112, 1988. 34

[37] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991. 34

[38] Dan Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013. URL http://hdl.handle.net/10062/29041. 273

[39] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, pages 192–206, 2008. 7, 273

[40] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis - (short paper). In *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, pages 57–64, 2012. 1, 7

[41] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography and Data Security, 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers*, pages 325–343, 2009. 1, 7

[42] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1175–1191, 2017. 11

[43] Gabriel Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 154–162, 1984. 280, 287

[44] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, pages 645–677, 2017. 5, 11, 156

[45] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In *Advances in Cryptology*

- CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings, pages 51–70, 2011. 166

[46] Megha Byali, Arun Joseph, Arpita Patra, and Divya Ravi. Fast secure computation for small population over the internet. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 677–694, 2018. 10, 105

[47] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. FLASH: fast and robust framework for privacy-preserving machine learning. *IACR Cryptology ePrint Archive*, 2019:1365, 2019. 7

[48] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 88–97, 2002. 274

[49] R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute, Israel, 1995. xii, xvi, 277, 281, 282, 283, 299

[50] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000. 22

[51] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 42–51, 1993. 274, 280

[52] Nishanth Chandran, Juan A. Garay, Payman Mohassel, and Satyanarayana Vusirikala. Efficient, constant-round and actively secure MPC: beyond the three-party case. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 277–294, 2017. 34, 103, 104, 132

[53] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. ASTRA: high throughput 3pc over rings with application to secure prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, CCSW@CCS 2019, London, UK, November 11, 2019*, pages 81–92, 2019. 1, 7, 11

[54] David Chaum. The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 591–602, 1989. 158, 225

[55] David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, pages 87–119, 1987. 1, 19

[56] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 11–19, 1988. 1, 6, 11, 156

[57] Koji Chida, Gembu Morohashi, Hitoshi Fuji, Fumihiko Magata, Akiko Fujimura, Koki Hamada, Dai Ikarashi, and Ryuichi Yamamoto. Implementation and evaluation of an efficient secure computation system using 'R' for healthcare statistics. *Journal of the American Medical Informatics Association*, 2014. 7

[58] Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 513–530, 2014. 8, 36, 104

[59] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 383–395, 1985. 16, 278

[60] Arka Rai Choudhuri, Michele Ciampi, Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Round optimal secure multiparty computation from minimal assumptions. Cryptology ePrint Archive, Report 2019/216, 2019. xi, 5, 13, 159, 160, 165, 169, 211, 220, 221, 222

[61] Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous MPC with linear communication complexity. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking, ICDCN 2015, Goa, India, January 4-7, 2015*, pages 5:1–5:10, 2015. 274

[62] Ashish Choudhury and Arpita Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Trans. Inf. Theory*, 63(1):428–468, 2017. 18, 275, 307, 308, 309, 310, 311, 312

[63] Ashish Choudhury, Martin Hirt, and Arpita Patra. Asynchronous multiparty computation with linear communication complexity. In *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, pages 388–402, 2013. 18, 275

[64] Michele Ciampi and Rafail Ostrovsky. Four-round secure multiparty computation from general assumptions. Cryptology ePrint Archive, Report 2019/214, 2019. 5, 162, 182, 222, 223

[65] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 364–369, 1986. iv, 6, 7, 11, 13, 156

[66] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, pages 466–485, 2014. 6, 22, 35, 71

[67] Ran Cohen, Iftach Haitner, Eran Omri, and Lior Rotem. Characterization of secure multiparty computation without broadcast. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, pages 596–616, 2016. 7, 8, 9, 10, 35, 71, 105, 109, 126

[68] Jeffrey Considine, Matthias Fitzi, Matthew K. Franklin, Leonid A. Levin, Ueli M. Maurer, and David Metcalf. Byzantine agreement given partial broadcast. *J. Cryptology*, 18(3): 191–217, 2005. 275

[69] Ronald Cramer, Ivan Damgård, and Stefan 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*, pages 325–334, 2000. 16

[70] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 342–362, 2005. 106

[71] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 141–150, 1998. 30

[72] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 572–590, 2007. 11, 103, 156

[73] Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 558–576, 2010. 11, 103, 156

[74] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *Information Security and Privacy, 12th Australasian Conference, ACISP 2007, Townsville, Australia, July 2-4, 2007, Proceedings*, pages 416–430, 2007. 11

[75] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*, pages 160–179, 2009. 2, 273, 274

[76] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012. 103

[77] Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential benchmarking based on multiparty computation. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, pages 169–187, 2016. 1

[78] Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, and Michael A. Raskin. On the communication required for unconditionally secure multiplication. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 459–488, 2016. 19

[79] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *J. ACM*, 40(1):17–47, 1993. 225, 274

[80] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, pages 164–181, 2011. 166

[81] Cynthia Dwork and Moni Naor. Zaps and their applications. *SIAM J. Comput.*, 36(6): 1513–1543, 2007. 34, 218

[82] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 308–317, 1990. 34

[83] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 427–437, 1987. 272, 274

[84] Matthias Fitzi. *Generalized communication and security models in Byzantine agreement*. PhD thesis, ETH Zurich, Zürich, Switzerland, 2003. 275

[85] Matthias Fitzi and Ueli 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, 2000. 275

[86] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, pages 121–136, 1998. 225

[87] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. General adversaries in unconditional multi-party computation. In *Advances in Cryptology - ASIACRYPT '99, International*

Conference on the Theory and Applications of Cryptology and Information Security, Singapore, November 14-18, 1999, Proceedings, pages 232–246, 1999. 225

[88] Matthias Fitzi, Martin Hirt, Thomas Holenstein, and Jürg Wullschleger. Two-threshold broadcast and detectable multi-party computation. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 51–67, 2003. 225

[89] Matthias Fitzi, Thomas Holenstein, and Jürg Wullschleger. Multi-party computation with hybrid security. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 419–438, 2004. 225

[90] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 191–219, 2015. 26, 38, 40

[91] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, pages 225–255, 2017. 7, 103, 104

[92] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 588–599, 2017. 6, 13, 159

[93] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 468–499, 2018. 5, 6, 13, 15, 16, 159, 160, 161, 169, 170, 178, 179, 181, 226, 227, 236, 237, 239, 240, 243, 252, 253, 254, 255, 256

[94] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *Theory of Cryptography - 11th Theory of Cryp-*

tography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings, pages 74–94, 2014. 6, 11, 13, 103, 156, 159

[95] Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 448–476, 2016. 5, 12, 13, 159, 165, 174

[96] Martin Geisler. Viff: Virtual ideal functionality framework, 2007. 7

[97] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 495–504, 2014. 212

[98] Daniel Genkin, Yuval Ishai, and Mor Weiss. Binary AMD circuits from secure multiparty computation. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, pages 336–366, 2016. 212

[99] Daniel Genkin, S. Dov Gordon, and Samuel Ranellucci. Best of both worlds in secure computation, with low communication overhead. In *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, pages 340–359, 2018. 158, 320

[100] Rosario Gennaro, Michael O. Rabin, and Tal 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, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, pages 101–111, 1998. 272

[101] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal 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*, pages 580–589, 2001. 16, 18, 19, 33, 276, 299, 302, 303, 321

[102] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual In-*

ternational Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings, pages 178–193, 2002. 6, 7, 8, 12, 13, 14, 15, 19, 33, 34, 41, 159, 163, 170, 248

[103] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 60(3):592–629, 2000. 37, 101

[104] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001. 22

[105] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004. 251

[106] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986. 29

[107] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or 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, 1987. 1, 11, 16, 103, 156

[108] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 63–82, 2015. 6, 8, 12, 13, 14, 15, 16, 22, 34, 41, 159, 163, 170, 226, 243, 248, 253, 254

[109] S. Dov Gordon, Samuel Ranellucci, and Xiao Wang. Secure computation with low communication from cross-checking. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, pages 59–85, 2018. 104, 105

[110] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, pages 308–326, 2010. 166

[111] Vipul Goyal, Silas Richelson, Alon Rosen, and Margarita Vald. An algebraic approach to non-malleability. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 41–50, 2014. 218

[112] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 132–150, 2011. 5, 8, 12, 13, 16, 34, 35, 63, 159, 174, 175, 179, 180, 226

[113] Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 488–520, 2018. xi, xv, 5, 11, 13, 156, 159, 160, 169, 211, 212, 213, 214, 215, 216, 217, 218, 220

[114] Shai Halevi, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. Best possible information-theoretic MPC. In *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part II*, pages 255–281, 2018. 158

[115] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Round-optimal fully black-box zero-knowledge arguments from one-way permutations. In *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I*, pages 263–285, 2018. x, 162, 189, 190, 201, 202, 203, 207, 209, 210

[116] Martin Hirt and Ueli M. Maurer. Robustness for free in unconditional multi-party computation. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 101–118, 2001. 239

[117] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 463–482, 2006. 274

[118] Martin Hirt, Ueli M. Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In *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*, pages 143–161, 2000. 239

[119] Martin Hirt, Ueli M. Maurer, and Vassilis Zikas. MPC vs. SFE : Unconditional and computational security. In *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*, pages 1–18, 2008. 158, 225

[120] Martin Hirt, Christoph Lucas, Ueli Maurer, and Dominik Raub. Graceful degradation in multi-party computation (extended abstract). In *Information Theoretic Security - 5th International Conference, ICITS 2011, Amsterdam, The Netherlands, May 21-24, 2011. Proceedings*, pages 163–180, 2011. 158, 225

[121] Martin Hirt, Christoph Lucas, Ueli Maurer, and Dominik Raub. Passive corruption in statistical multi-party computation - (extended abstract). In *Information Theoretic Security - 6th International Conference, ICITS 2012, Montreal, QC, Canada, August 15-17, 2012. Proceedings*, pages 129–146, 2012. 158

[122] Martin Hirt, Christoph Lucas, and Ueli Maurer. A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 203–219, 2013. 13, 158, 225, 227, 233, 236

[123] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 650–662, 2014. 38

[124] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 483–500, 2006. 11, 157, 158, 225

[125] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 572–591, 2008. 103

[126] Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 577–594, 2010. 6, 12, 34

[127] Yuval Ishai, Jonathan Katz, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On achieving the "best of both worlds" in secure multiparty computation. *SIAM J. Comput.*, 40(1):122–141, 2011. 11, 157, 158, 174, 225, 229, 250, 252

[128] Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. Identifying cheaters without an honest majority. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 21–38, 2012. 160, 166

[129] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 359–378, 2015. 6, 7, 8, 9, 10, 12, 15, 33, 34, 35, 36, 64, 103, 104, 105, 106, 112, 121, 132, 318, 319

[130] Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 430–458, 2016. 35, 63, 71, 160, 165, 250

[131] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 1989. 106

[132] Zahra Jafargholi and Daniel Wichs. Adaptive security of yao's garbled circuits. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, pages 433–458, 2016. 31

[133] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 955–966, 2013. 26, 38, 40

[134] Jonathan Katz. On achieving the "best of both worlds" in secure multiparty computation. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 11–20, 2007. 11, 157, 225

[135] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(028), 2006. 16

[136] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, pages 335–354, 2004. 5, 34

[137] Jonathan Katz and Ji Sun Shin. Modeling insider attacks on group key-exchange protocols. In *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*, pages 180–189, 2005. 29

[138] Jonathan Katz, Steven Myers, and Rafail Ostrovsky. Cryptographic counters and applications to electronic voting. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, pages 78–92, 2001. 11

[139] Jonathan Katz, Chiu-Yuen Koo, and Ranjit Kumaresan. Improving the round complexity of VSS in point-to-point networks. *Inf. Comput.*, 207(8):889–899, 2009. 18, 286, 307, 308, 311, 314

[140] Mehmet S Kiraz and Berry Schoenmakers. A protocol issue for the malicious case of yao's garbled circuit construction. In *27th Symposium on Information Theory in the Benelux*, 2006. 39

[141] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *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*, pages 486–498, 2008. 85, 135

[142] Dror Lapidot and Adi Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, pages 353–365, 1990. 221

[143] John Launchbury, Iavor S. Diatchki, Thomas DuBuisson, and Andy Adams-Moran. Efficient lookup-table protocol in secure multiparty computation. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 189–200, 2012. 7

[144] John Launchbury, Dave Archer, Thomas DuBuisson, and Eric Mertens. Application-scale secure multiparty computation. In *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 8–26, 2014. 1, 7

[145] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 1–17, 2013. 36, 39, 55, 68, 103

[146] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. 2017. 22

[147] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, pages 52–78, 2007. 39, 40, 56, 90, 103

[148] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009. 31

[149] Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. *J. Priv. Confidentiality*, 1(1), 2009. 1

[150] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *J. Cryptology*, 25(4):680–722, 2012. 103

[151] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 319–338, 2015. 103

[152] Christoph Lucas, Dominik Raub, and Ueli M. Maurer. Hybrid-secure MPC: trading information-theoretic robustness for computational privacy. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 219–228, 2010. 11, 12, 157, 158, 225

[153] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996. 41, 275

[154] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977. 281, 282, 293

[155] Eleftheria Makri, Dragos Rotaru, Nigel P. Smart, and Frederik Vercauteren. EPIC: efficient private image classification (or: Learning from the masters). In *Topics in Cryptology - CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings*, pages 473–492, 2019. 1, 7

[156] Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, pages 458–473, 2006. 39

[157] Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52, 2018. 1, 7, 11

[158] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38, 2017. 1, 7, 11

[159] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 591–602, 2015. xvii, 7, 8, 9, 10, 26, 34, 35, 36, 103, 104, 105, 107, 111, 121, 126, 131, 132, 133, 134, 319

[160] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 735–763, 2016. 6, 13, 159, 192

[161] Divya G. Nair, V. P. Binu, and G. Santhosh Kumar. An improved e-voting scheme using secret sharing based secure multi-party computation. *CoRR*, abs/1502.07469, 2015. 11

[162] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991. 30

[163] Jesper Buus Nielsen and Claudio Orlandi. Cross and clean: Amortized garbled circuits with constant overhead. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, pages 582–603, 2016. 103

[164] Rafail Ostrovsky, Alessandra Scafuro, Ivan Visconti, and Akshay Wadia. Universally composable secure computation with (malicious) physically uncloneable functions. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 702–718, 2013. 166

[165] Arpita Patra and C. Pandu Rangan. Communication and round efficient information checking protocol. *CoRR*, 2010. URL `http://arxiv.org/abs/1004.3504`. 160, 166

[166] Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 425–458, 2018. 6, 8, 12, 13, 14, 15, 16, 35, 104, 159, 163, 170, 226, 248, 249

[167] Arpita Patra and Divya Ravi. On the power of hybrid networks in multi-party computation. *IEEE Trans. Inf. Theory*, 64(6):4207–4227, 2018. 2, 19, 276

[168] Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. Cryptology ePrint Archive, Report 2018/481, 2018. 6, 7, 8, 163, 248

[169] Arpita Patra and Divya Ravi. Beyond honest majority: The round complexity of fair and robust multi-party computation. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, pages 456–487, 2019. 15, 16, 158, 226

[170] Arpita Patra and Divya Ravi. Beyond honest majority: The round complexity of fair and robust multi-party computation. Cryptology ePrint Archive, Report 2019/998, 2019. 44, 48

[171] Arpita Patra and Ajith Suresh. BLAZE: blazing fast privacy-preserving machine learning. *IACR Cryptology ePrint Archive*, 2020:42, 2020. 7

[172] Arpita Patra, Ashish Choudhary, Tal Rabin, and C. Pandu Rangan. The round complexity of verifiable secret sharing revisited. In *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 487–504, 2009. 15, 33, 41

[173] Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. Simple and efficient asynchronous byzantine agreement with optimal resilience. In *Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing, PODC 2009, Calgary, Alberta, Canada, August 10-12, 2009*, pages 92–101, 2009. 160, 166, 274

[174] Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous verifiable secret sharing and multiparty computation. *J. Cryptology*, 28(1):49–109, 2015. 274

[175] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 129–140, 1991. 30, 228, 234, 272, 274

[176] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 554–571, 2008. 159

[177] Tal Rabin and Michael 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, 1989. 11, 103, 156, 274

[178] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, pages 371–388, 2004. 106, 107

[179] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology - CRYPTO*

2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings, pages 566–598, 2001. 159

[180] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979. 30, 233

[181] Abhi Shelat and Chih-Hao Shen. Fast two-party secure computation with minimal assumptions. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 523–534, 2013. 40, 56

[182] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982. 1, 27, 40, 85, 103, 135

[183] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 220–250, 2015. 85, 131, 135